NAVAL POSTGRADUATE SCHOOL

MONTEREY, CALIFORNIA

THESIS

SENSORS AND ALGORITHMS FOR AN UNMANNED SURF-ZONE ROBOT

by

Oscar García

December 2015

Thesis Advisor: Richard Harkins
Second Reader: Peter Crooker

Approved for public release; distribution is unlimited

THIS PAGE INTENTIONALLY LEFT BLANK

REPORT DOCUMENTATION PAGE			Form Approved OMB No. 0704–0188
Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instruction, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188) Washington, DC 20503.			
1. AGENCY USE ONLY *(Leave blank)*	**2. REPORT DATE** December 2015	**3. REPORT TYPE AND DATES COVERED** Master's thesis	
4. TITLE AND SUBTITLE SENSORS AND ALGORITHMS FOR AN UNMANNED SURF-ZONE ROBOT			**5. FUNDING NUMBERS**
6. AUTHOR(S) Oscar García			
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey, CA 93943-5000			**8. PERFORMING ORGANIZATION REPORT NUMBER**
9. SPONSORING /MONITORING AGENCY NAME(S) AND ADDRESS(ES) N/A			**10. SPONSORING / MONITORING AGENCY REPORT NUMBER**
11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government. IRB Protocol number ____N/A____.			
12a. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public release; distribution is unlimited			**12b. DISTRIBUTION CODE**

13. ABSTRACT (maximum 200 words)

The design, construction, integration and implementation of electronics, sensors, actuators and power supplies for a surf-zone autonomous vehicle are presented. Physical models and lab-test characterizations are used to address limitations and achieve improved performance through signal-processing techniques. A deterministic centralized pooling-communication protocol is designed and implemented for use over a network of microcomputers and microprocessors with limited computational resources. A series of algorithms are developed to achieve autonomy over land and at sea. Autonomy functions include waypoint navigation, obstacle avoidance, sea-to-land transition, operation environment detection, depth maintenance and wireless communications—all of which support basic autonomous intelligence, surveillance and reconnaissance missions for missions over a beach front.

14. SUBJECT TERMS robotics, unmanned systems, virtual potential filed, inertial measurement unit, pressure sensors, motor control, microprocessors, Kalman filter			**15. NUMBER OF PAGES** 211
			16. PRICE CODE
17. SECURITY CLASSIFICATION OF REPORT Unclassified	**18. SECURITY CLASSIFICATION OF THIS PAGE** Unclassified	**19. SECURITY CLASSIFICATION OF ABSTRACT** Unclassified	**20. LIMITATION OF ABSTRACT** UU

NSN 7540–01-280-5500 Standard Form 298 (Rev. 2–89)
Prescribed by ANSI Std. 239–18

THIS PAGE INTENTIONALLY LEFT BLANK

Approved for public release; distribution is unlimited

SENSORS AND ALGORITHMS FOR AN UNMANNED SURF-ZONE ROBOT

Oscar García
Lieutenant Commander, Chilean Navy
B.S., Academia Politécnica Naval, 2002

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN APPLIED PHYSICS

from the

NAVAL POSTGRADUATE SCHOOL
December 2015

Approved by: Richard Harkins
Thesis Advisor

Peter Crooker
Second Reader

Kevin Smith
Chair, Department of Physics

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

The design, construction, integration and implementation of electronics, sensors, actuators and power supplies for a surf-zone autonomous vehicle are presented. Physical models and lab-test characterizations are used to address limitations and achieve improved performance through signal-processing techniques. A deterministic centralized pooling-communication protocol is designed and implemented for use over a network of microcomputers and microprocessors with limited computational resources. A series of algorithms are developed to achieve autonomy over land and at sea. Autonomy functions include waypoint navigation, obstacle avoidance, sea-to-land transition, operation environment detection, depth maintenance and wireless communications—all of which support basic autonomous intelligence, surveillance and reconnaissance missions for missions over a beach front.

THIS PAGE INTENTIONALLY LEFT BLANK

TABLE OF CONTENTS

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF FIGURES

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF TABLES

LIST OF ACRONYMS AND ABBREVIATIONS

APCDU	Aft Power Converter & Distribution Unit
ARS	Attitude – Reference – System
ASCII	American Standard Code for Information Interchange
AXV	Surf Zone Autonomous Vehicle
A2/AD	Anti-Access and Area Denial
CEP	Circular Error Probability
CF	Complementary Filter
CW	Continuous Wave
CWMD	Counter-Weapons of Mass Destruction
GGA	Global Positioning System Fix Data
COTS	Commercial Off The Shelf
DGPS	Differential GPS
DOD	Department of Defense
DR	Dead reckoning
DUT	Device Under Test
EGNOS	European Geostationary Navigation Overlay Service
EKF	Extended Kalman Filter
ESC	Electronic Speed Control
FPCDU	Forward Power Converter & Distribution Unit
GPS	Global Positional System
GUI	Graphic User Interface
IMU	Inertial Measurement Unit
IR	Infra-Red
IRSIC	IR Switch Integrator Circuit
ISR	Intelligence, Surveillance and Reconnaissance
LKF	Linear Kalman Filtering
LPF	Low Pass Filter
LSB	Least Significant Bit
MEMS	Micro Electrical Mechanical System

NMEA	National Marine Electronic Association
MOSARt	Mobil Surf-Zone Amphibious Robot
MP	Main Processor
MTAIC	Multi Tone Active Interference Canceller
PDB	Power Distribution Board
PID	Proportional – Integral – Derivative
PP	Preprocessor
PWM	Pulse Width Modulated
RCP	Right-hand Circular Polarized antenna
RF	Radio Frequency
RM	Recommended Minimum
VPF	Virtual Potential Field
SA	Sense-Act
SSH	Secure Shell
SW	Software
MOSARt	Surf-Zone Prototype 1
UW	Underwater
RTC	Real Time Clock
VPN	Virtual Private Network
WAAS	Wide Area Augmentation System
WP	Waypoint

ACKNOWLEDGMENTS

I would like to thank the Chilean Navy and the United States Navy for giving me the opportunity to be part of this excellent academic program.

To all my professors of the Physics Department: It was an (extremely!) hard but enriching experience. I felt very motivated during all the courses, and I am very grateful for the 24/7 disposition that you all had with all the students.

To Steve Jacobs, thanks for all your advice and support.

To Peter Crooker, thanks for sharing all your knowledge with me.

To my thesis advisor, Professor Richard Harkins: Thanks for all the advice, guidance, trust and support. Thanks for translating my thesis form "Spanglish" to English!

To my classmates from the U.S. and from all over the world: I take nothing but the best impression from your countries. Thanks for making me feel at home.

To Manuel Ariza, from the Colombian Navy, the man in charge of the structural and mechanics parts of the robot, it was a pleasure to work together.

To my family back in Chile, thanks for your constant support through daily emails and messages (mother!), and to my father for all his wise advice.

To my 5-year-old, Cristobal, thanks for never accepting an "I am studying!" as an excuse to avoid playing with you. I would not trade those memories for anything in the world!

To my beloved wife, Sarina. Thanks for sacrificing your career so we could embark together into this adventure. Thanks for playing the role of mother, father, housewife, teacher … and my shrink! Thanks for supporting all those lonely hours and all those hours that I was present in body, but my mind was wandering in the world of physics. I promise I'll do better in the future.

To God, thanks for everything!

THIS PAGE INTENTIONALLY LEFT BLANK

I. INTRODUCTION

A. MOTIVATION

In today's world, the role of unmanned systems in military missions is unquestionable. As indicated in [1]: unmanned systems "enhance situational awareness, reduce human workload, improve mission performance, and minimize overall risk to both civilian and military personnel". Consequently, the Department of Defense (DOD) has identified several important autonomous unmanned missions that are related to:

1. Intelligence, surveillance and reconnaissance (ISR)
2. Counterterrorism
3. Counter-weapons of mass destruction (CWMD)
4. Multi-disciplinary operations, including anti-access and area denial (A2/AD) [1]

IRS and a subset of the A2/AD mission is considered the "hostile surf-zone" for this project. We explore an Autonomous Vehicle (AXV), as an unmanned robot, that has the capability to reach the beachfront from the sea and perform ISR missions and A2/AD.

Considering the hostile A2/AD environment, chances of post-mission survival or recovery are low. Therefore, the word "expendable" and "low cost" are concepts that are closely related with the design of an AXV (from now on MObile Surf-zone Amphibious Robot – MOSARt).

B. OBJECTIVES

The MOSARt concept has been explored and studied by the Physics Department over the last eight years, focusing primarily on platforms that operate on sandy terrain in a manual or semi-autonomous mode. Different sensors and control algorithms have been tested separately, but without the constraints imposed by the maritime environment. For this thesis, an integrated electronic sensor/ control and processing assembly is proposed, along with the power bus

and power supply needed for a multi-environment operation. The main operational tasks of these implementations are:

1. At sea (underwater (UW) environment):

    a. Motor control and thruster to maintain a desired course and depth.
    b. Dead reckoning (DR).
    c. Sea to land transition mobility.

2. On the beachfront:

    a. Active and passive/ inertial positioning.
    b. Waypoint (WP) navigation (motor control).
    c. Obstacle avoidance in an efficient and simple manner.
    d. Tail deployment.
    e. Wireless communications.

C. EXCLUSIONS

Structural design of the body, wheels, thrusters and motors of MOSARt are not part of the scope of this thesis. Nevertheless, because the design and construction of MOSARt was done in parallel to this study (Ariza [2]), structure was considered for the sensors and electronics design and implementation. A high level of coordination and team-work was required.

MOSARt design was governed by specific operational parameters, as follows:

1. Sea state 2 (0.1 to 0.5 m waves), landing on a beach type 2 (perpendicular wave approach), angle of incidence 1˚ - 10˚ (with respect the normal to the coastline)

2. Day - night operation

3. Sandy beach, with occasional rocks or other types of obstacles (cliff, holes, rocks, animals (dead or alive))

4. GPS coverage not guaranteed (due to trees, etc.)

II. MOSART COMPONENTS DESCRIPTIONS

A. MOSART STRUCTURE GENERAL DESCRIPTION.

The general mechanical design (structure, thrusters and motors) are discussed by Ariza in [2]. Figure 1 shows a general view, along with overall dimensions and weight. The main body, sensor array suit, tail structure and Whegs are 3D printed material. The chassis is made of aluminum. Inside the main body, a 30.5 cm x 17.8 cm watertight cylinder (intended for power supply distribution and main electronics) is placed longitudinal to the main structure.

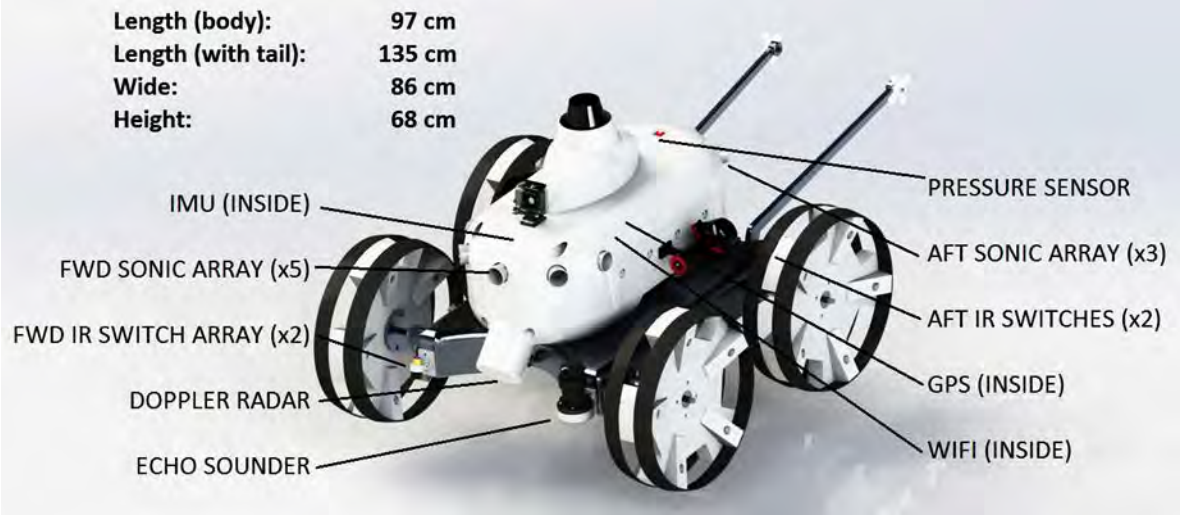


Figure 1 Rendered Image of MOSARt General Structure.

Adapted from [2]: M. Ariza, *The Design and Implementation of a Prototype Surf zone Robot for Waterborne Operations*. Monterey, CA: NPS, 2015.

MOSARt uses 3 thrusters: 2 along each side for forward/ reverse motion. Port and starboard turns are accomplished by applied torque of each thruster over the structure of the robot. A central thruster is shown on the top of the robot, used for up/ down movements.

Four Whegs (modified wheels) are implemented, driven by 2 brushed DC motors (port and starboard). The Whegs are an adaptation made in [2] to the

design defined on [3]. Forward and aft Whegs are connected by a transmission chain.

The tail is used for stability support during land operations. The design and control of the tail is an adaptation of previous thesis [4].

No underwater control fins are used. Buoyancy and underwater stability is done through manipulation of the center of mass.

B. GENERAL COMPONENTS SELECTION

Taking into consideration the tasks described in I.B and the general structure to control, described in A, different requirements were identified and associated with physical sensing capabilities. The results are listed in Table A1 in Appendix A. The specific hardware selections are detailed in Table A2 of the same appendix.

C. HARDWARE DESCRIPTION

The following is a discussion of the selected hardware used in the project. A brief physics-related explanation is explored and then related to the integration, limitations and testing process.

1. Adafruit 10-Degrees Of Freedom (DOF) Inertial Measurement Unit (IMU)

a. General Description

This unit, shown in Figure 2, combines 4 Micro Electro-Mechanical System (MEMS) units:

1. LSM303DLH 3-axis accelerometer: ±2g/±4g/±8g/±16g selectable scale
2. L3GD20H 3-axis gyroscope: ±250, ±500, or ±2000 degree-per-second scale
3. LSM303DLH 3-axis compass: ±1.3 to ±8.1 gauss magnetic field scale (incorporated on the same chip as the accelerometer)

4. BMP180 barometric pressure/temperature: -40 to 85 °C, 300 - 1100hPa range, 0.17m resolution

Figure 2 Adafruit 10 DOF IMU.

The IMU is packaged in a 38 mm x 23 mm x 3 mm unit. Having all four sensors packaged in a compact form avoids dealing with parallax alignment issues between sensors. Each component uses Inter Integrated Circuit (I2C) protocol to output raw data. It is 5 v and 3.3 v compatible. For MOSARt application, only the accelerometer, gyroscope and the magnetic compass units are used.

b. MEMS Accelerometer Principle of Operation

This MEMS device is designed to measure linear acceleration, which includes gravity, in the three axes. If the accelerometer is mounted on a rigid body, and the external forces acting on the sensor are negligible in comparison to the normal forces of gravity, the raw output of the accelerometer would be a unit vector representing the acceleration of gravity. This raw output can be modeled as [5]:

$$\vec{a}_m = -\frac{\vec{F}_g}{m} \tag{1}$$

In the previous equation, $\vec{a}_m$ is the measured acceleration (x, y, z), m is the mass of the body and $\vec{F}_g$ is the force due gravity expressed in the body

frame. If only one axis is considered, equation (1) can be represented by a cantilever with a proof mass on its tip (see Figure 3).

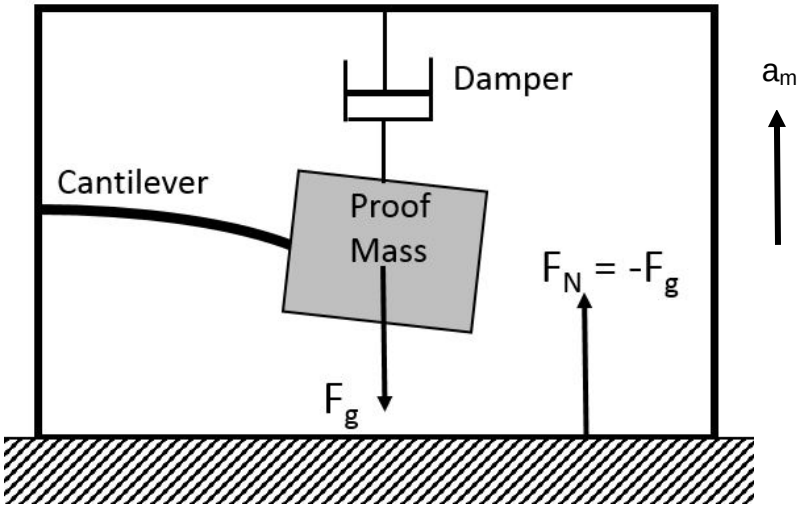


Figure 3 Simple Representation of 1-Axis Accelerometer.

When the accelerometer is static, F_g produces a downward deflection of the "proof mass" that is proportional to an acceleration of the sensor at the rate of gravity. The measurement of this deflection constitutes a signal proportional to the corresponding axis acceleration. In this scenario, the Euler angles *Pitch* and *Roll* can be calculated by first considering that the output of the accelerometer (a_m) corresponds to a unity gravity vector *g* that points upwards through the z axis (as it is calculaded with the normal force $\vec{F}_N$ that prevent the sensor from accelerating toward the center of the earth), rotated about the x axis (*Roll*: ϕ), y axis (*Pitch*: θ) and the z axis (*Yaw*: ψ):

$$a_m = \begin{pmatrix} a_{mx} \\ a_{my} \\ a_{mz} \end{pmatrix} = R\vec{g} = R \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix} g \qquad (2)$$

R represent the rotation matrices on the three axes, which relates the angular rotations on the robot body frame with the Earth's gravitational vector.

The order that this rotation matrices are applied does not commute. If the Aerospace Rotation Sequence R_{xyz} [6] is selected, equation (2) becomes:

$$a_m = R_{x(\phi)} R_{y(\theta)} Rz_{(\psi} \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix} g$$

$$= \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos\phi & \sin\phi \\ 0 & -\sin\phi & \cos\phi \end{pmatrix} \begin{pmatrix} \cos\theta & 0 & -\sin\theta \\ 0 & 1 & 0 \\ \sin\theta & 0 & \cos\theta \end{pmatrix} \begin{pmatrix} \cos\psi & \sin\psi & 0 \\ -\sin\psi & \cos\psi & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix} g \qquad (3)$$

$$a_m = \begin{pmatrix} -\sin\theta \\ \cos\theta\sin\phi \\ \cos\theta\cos\phi \end{pmatrix} g$$

The result of (3) can be related to the normalized value of the output vector of the accelerometer:

$$\frac{\vec{a}_m}{\|\vec{a}_m\|} = \frac{1}{\sqrt{a_{mx}^2 + a_{my}^2 + a_{mz}^2}} \begin{pmatrix} a_{mx} \\ a_{my} \\ a_{mz} \end{pmatrix} = \begin{pmatrix} -\sin\theta \\ \cos\theta\sin\phi \\ \cos\theta\cos\phi \end{pmatrix} \qquad (4)$$

Solving for roll and pitch in (4):

$$Roll = \phi = \tan 2^{-1} \left(\frac{a_{my}}{a_{mz}} \right) \qquad (5)$$

$$Pitch = \theta = \tan^{-1} \left(\frac{-a_{mx}}{a_{my}\sin\phi + a_{mz}\cos\phi} \right) = \tan^{-1} \left(\frac{-a_{mx}}{\sqrt{a_{my}^2 + a_{mz}^2}} \right) \qquad (6)$$

The "*tan2*$^{-1}$" corresponds to the arctangent function, but it takes into consideration the position of the minus sign, if any. Note that an accelerometer does not have the capability to measure the *Yaw* angle, as it is insensitive to rotations about the gravitational field vector (this will be addressed with gyroscope and magnetometer).

If non-trivial external forces are acting over the sensor (others than gravity), then equation (1) must be modified:

$$\overrightarrow{a_m} = \frac{1}{m}\left(\vec{F} - \vec{F}_g\right) \tag{7}$$

$\vec{F}$ represents the sum of all forces on the body (including gravity), expressed in the sensor body frame. Any acceleration perpendicular to the cantilever will cause a deflection of the proof mass. This deflection is proportional to the acceleration (for deflections below the operational range limits). Equation (7) and Figure 3 indicate that the MEMS accelerometers measure gravity by sensing the normal forces ($\vec{F}_N$) that prevent the sensor from accelerating toward the center of the earth.

The external forces (besides gravity) acting on the sensor constitute an important limitation to the application of equations (5) and (6). In This case, $\vec{F}$ will be composed not only by $\vec{F}_N$ and $\vec{F}_g$, therefore the extraction of the normal force (needed for pitch and roll calculations) as shown in equation (7), is not going to be possible. Also, the result of equation (3) will not be valid, affecting the roll and pitch calculation derived on (5) and (6).

For a less idealized model of the accelerometer, scale factors, output bias and cross axis alignments must be taken into account. This results in additional errors in the output of the sensor. Incorporating these error sources in (7), a more realistic model is obtained [5]:

$$\vec{a}_m = M_a\left[\frac{1}{m}S_a(T)(\vec{F} - \vec{F}_g) - \beta_a(T) + \vec{\zeta}\right] \tag{8}$$

M_a is the sensor's misalignment matrix. It describes the effects of cross axis misalignment. $\vec{\zeta}$ is the Zero-g level offset, and is an actual output signal when no acceleration is present. The cause of this is a temperature independent stress to the MEMS sensor. The other factors are temperature dependent and include: $\beta_a(T)$, which accounts for a vector of temperature-variant output biases and finally $S_a(T)$, the diagonal temperature dependent sensitivity matrix [5]:

$$S_a(T) = \begin{pmatrix} S_{ax}(T) & 0 & 0 \\ 0 & S_{ay}(T) & 0 \\ 0 & 0 & S_{az}(T) \end{pmatrix} \quad (9)$$

Therefore, for proper acceleration sensing, the term described by (7) must be extracted from (8). Solving for *(F – F_g)/m* yields the corrected acceleration output $\vec{a}_{mc}$:

$$\vec{a}_{mc} = S_a^{-1}(T)\left[M_a^{-1}\vec{a}_m + \beta a(T) - \vec{\zeta} \right] \quad (10)$$

The value of M_a and $\vec{\zeta}$ for the LSM303DLHC accelerometer are experimentally determined and factory-calibrated [7]. For the temperature dependent errors, the LSM303DLHC has a built in temperature sensor. The resolution of the sensor, i.e. the Least Significant Bit (LSB) value is 1 mg/LSB.

c. MEMS Rate Gyros Principle of Operation

Using the same methodology, the output of a three axis gyro can be modeled [5]:

$$\vec{\omega}_m = M_g\left[S_g(T)\vec{\omega} - \beta_g(T) + \vec{\zeta} + \mathrm{C_{g,a}}\vec{\mathrm{a}}_{\mathrm{mc}} \right] \quad (11)$$

$\vec{\omega}_m$ is the measured angular rate and $\vec{\omega}$ is a vector representing the actual body frame angular rates. All the rate readings are in [rad/s]. $\mathrm{C_{g,a}}$ is a matrix that accounts for the amount of coupling between the physical acceleration of the rate gyro and deviations in the output angular rate. This term is counteracted with the single drive mass design of the L3GD20H [8], as shown in Figure 4.

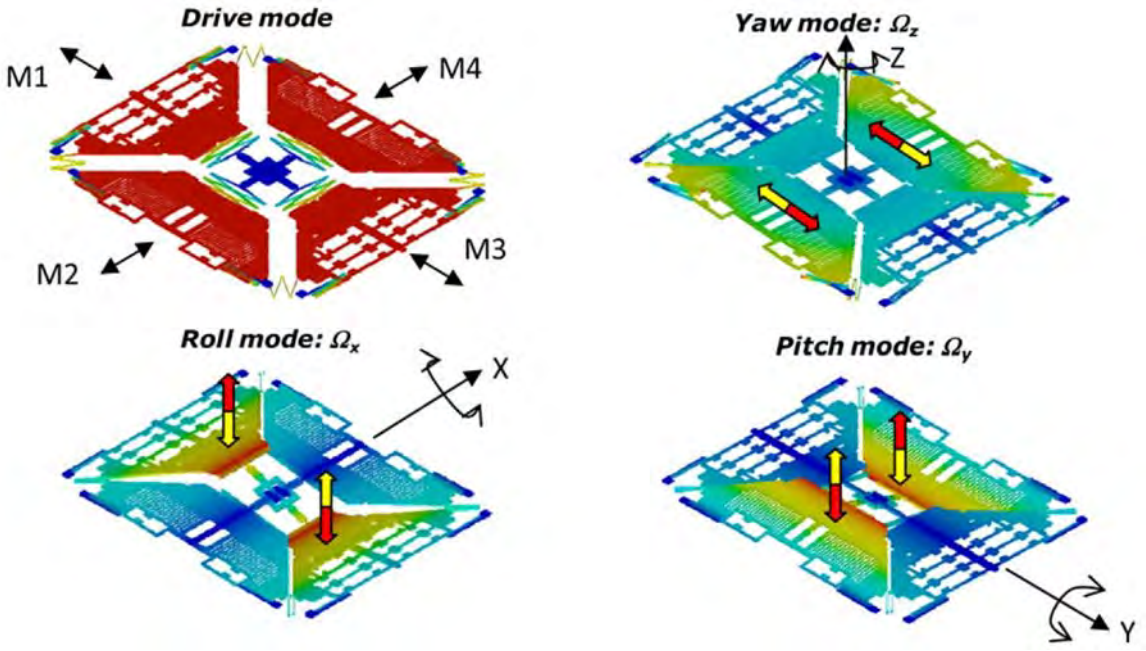


Figure 4 Single Driven Mass for the L3GD20H Gyroscope.

Source: [8]: ST Microelectronics, *TA0343 Technical Article: Everything About ST Microelectronics' 3-Axis digital MEMS Gyroscopes*. U.S.: ST Microelectronics, 2011.

For the L3GD20H, the information of interest is the actual angular rate of each axis. Solving for ω in equation (11) and neglecting $C_{g,a}$ gives:

$$\vec{\omega} = S_g^{-1}(T)\left[M_g^{-1}\vec{\omega}_m + \beta_g(T) - \zeta\right] \tag{12}$$

The main errors, sensitivity and zero-bias are compensated during factory calibration [9] and according to [8] the L3GD20H gives excellent performance regarding temperature dependent errors.

With the angular velocity for each axis, as long as the body frame coincide with the Euler angles' reference frame, it is possible to obtain the Euler angle during the rotation of the axe, from a pre-defined start angle α_i.

$$\alpha(t) = \alpha_i + \int_0^t \omega(t)dt \approx \alpha_i + \sum_0^t \omega(t)T_s \tag{13}$$

The integral in (13) is approximated by serial sum, as it is not possible to compute a continuous integral in a digital computer. In this last case, the parameter T_s becomes relevant, as it presents the sampling period. If the gyro output changes faster than T_s, the integral approximation will become less accurate, resulting on drift. This error will increase over time. During steady state scenarios, the gyro's error will also cause drift to appear, regardless of the sampling rate.

When the body frame does not coincide with the reference frame, before integration with respect to time, the angular velocities in the body frame have to be rotated to relate then to the Euler angles:

$$\begin{Bmatrix} Roll_{ang\_vel} \\ Pich_{ang\_vel} \\ Yaw_{ang\_vel} \end{Bmatrix} = \begin{Bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{Bmatrix} = \begin{bmatrix} 1 & \sin\phi\tan\theta & \cos\phi\tan\theta \\ 0 & \cos\phi & -\sin\phi \\ 0 & \sin\phi\sec\theta & \cos\phi\sec\theta \end{bmatrix} \begin{Bmatrix} p \\ q \\ r \end{Bmatrix} \tag{14}$$

In the above equation, *p*, *q* and *r* are the angular rates in each body frame. As (13), initial attitude values are needed.

d. MEMS Magnetometer Principle of Operations

These devices are based on a three-axis Hall effect sensor, used to measure the earth's magnetic field. Figure 5 shows a schematic of a Hall sensor. A constant current flow is applied to a semiconductor (Hall Element). When a magnetic $\vec{B}$ field passes through this element, the Lorentz force $\vec{F}$, shown in equation (15), acts on the charges *q* moving with a velocity $\vec{v}$ inside the semiconductor:

$$\vec{F} = q(\vec{E} + \vec{v} \times \vec{B}) \tag{15}$$

$\vec{E}$ is the electric field. As the second term of (15) is transverse to the velocity and the magnetic field, an output voltage can be sensed through electrodes at the transverse dimensions of the Hall Element. The Hall voltage (V_H) is related to the magnetic field by:

$$V_H = R_H \frac{IB}{t} \tag{16}$$

In the previous equation, R_H is the Hall Coefficient, I is the current across the semiconductor and t is the plate thickness. Note that to generate a potential difference, the magnetic field must be perpendicular to the flow current.

With a two axis magnetometer, is possible to measure the magnetic heading, by measuring the earth's horizontal magnetic field. A third axis is needed to account for off-axis sensor movement. For a more accurate result, an additional accelerometer is needed for gravity vector reference (this device is packed with the LSM303DLHC accelerometer). This compensation must be software implemented by the end user.

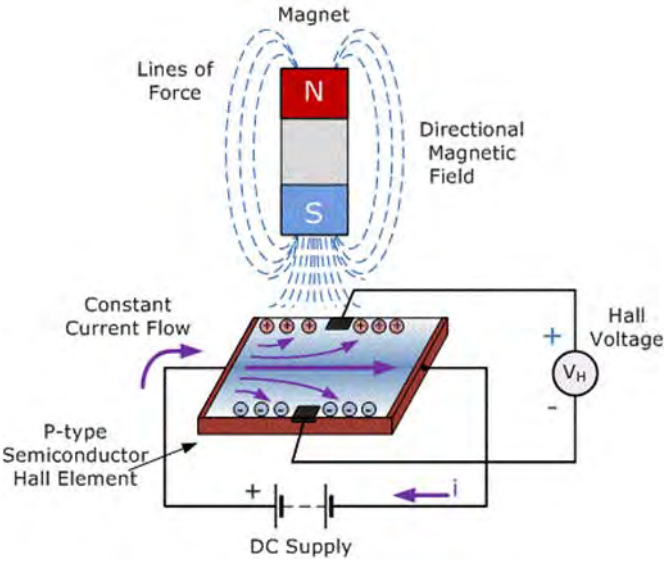


Figure 5 Representation of a Hall Sensor.

Source: [10]: W. Storr, *Electronic Tutorials: Hall Effect Sensor* [Online]. Available: http://www.electronics-tutorials.ws/electromagnetism/hall-effect.html

A model equivalent to the previous analysis can be described [5]:

$$\vec{b} = S_b^{-1}(T)\left[M_b^{-1}\vec{b}_m + \beta_b(T) \right] \tag{17}$$

In (17), $\vec{b}$ and $\vec{b}_m$ represent the actual and measured magnetic field (in μT) respectively. $S_b^{-1}(T)$ and M_b^{-1} are the inverse of the magnetometer sensitivity and misalignment matrix. $\beta_b(T)$ is the bias vector of the magnetometer. Tilt compensation is not included in this model.

Although error sources are compensated at the factory, the sensor is still effected by unwanted magnetic fields. These can include sources from nearby magnetic and ferrous metals, time variant electromagnetic waves and communications or power lines among others. All of them can distort the final heading output.

Local magnetic field distortions are attributed to either soft iron or hard iron sources. The first accounts for ferrous objects that bend the Earth's magnetic field. Hard iron describes the effects of objects that produce magnetic fields (magnets, motors, speakers, etc.). The result is a need for field compensation. A fully compensated output will represent readings over a 4π steradian as a perfect sphere centered on the origin. Soft iron distortions will cause the sphere to distort into an ellipse. Hard iron effects will shift the center of the sphere off axis.

e. IMU Bench Test

Figure 6 shows the Device Under Test (DUT). In this setup, an Arduino-UNO is used to power the IMU and receive data through its IC2 channel. An Attitude – Reference – System (ARS) library specially designed for this IMU was used: the *Adafruit_ARS*. This library was modified to output roll, pitch, heading and timestamp in the desired format and with a maximum refresh update rate at 100 Hz.

For the *Adafruit_LSM303DLHC* magnetometer calibration check, the "*magsensor"* program was modified to output the raw X, Y and Z measurements of the Hall sensors.

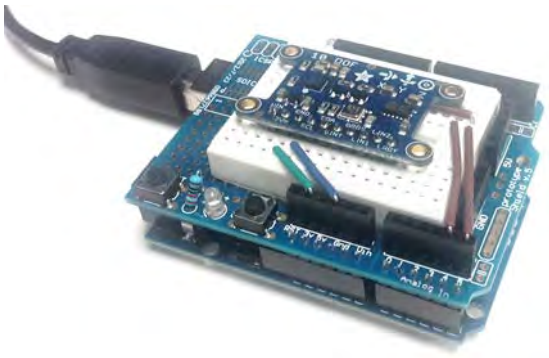


Figure 6 DUT Arduino-UNO and IMU Installed in a Proto-Shield.

For the magnetometer calibration check, the DUT was waived around by hand, through a 4π solid angle. The sole source of electromagnetic interference was caused by a wireless computer connection. Data from the X, Y and Z outputs where saved to a text file and displayed in Matlab. Figure 7 shows the results of this test.

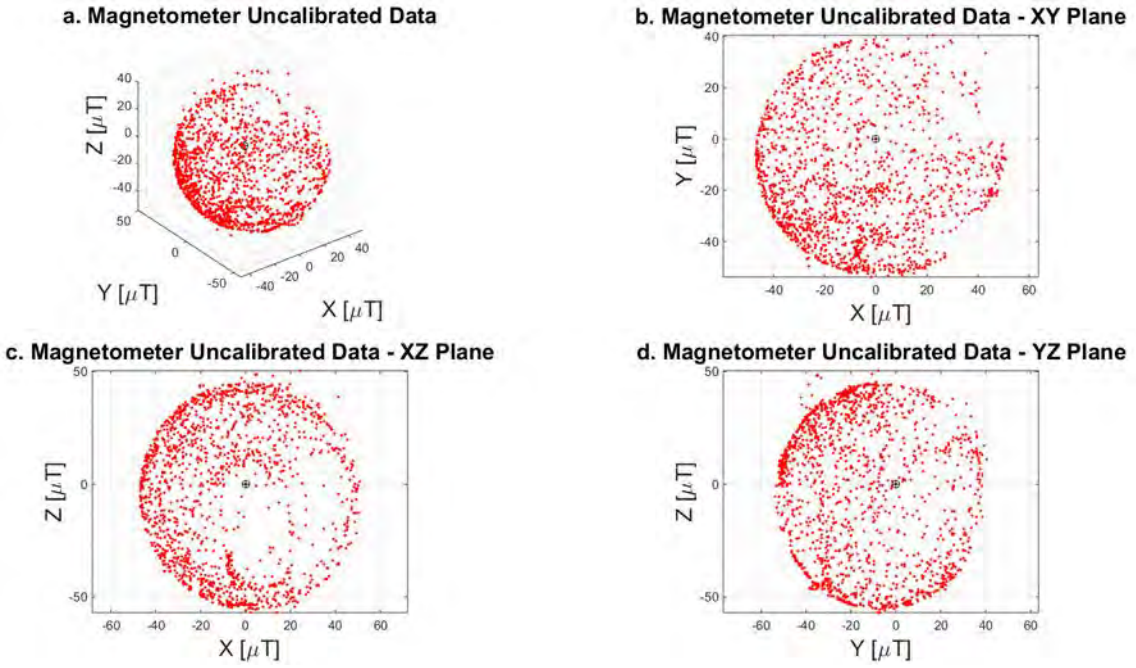


Figure 7 Un-Calibrated Magnetometer Output Results.

Figure 7.a shows a 3D view of the magnetometer output. Each dot represents a reading in X, Y, Z in μT. Figure 7.b to d are the same readings but referred to a 2D plane. Hard and soft iron effects are clearly seen: Soft iron de-calibration (sphere distortion) was caused by proximity of circuit boards. Offset of the center of the sphere, due to hard iron effects, was attributed to the wireless connection.

For attitude tests, the *KINOVA Jaco robotic arm* was used, to provide a predictable and repeatable motion. It is noted that the *Adafruit_ARS* library did not use sensor gyroscope information. Therefore pitch and roll corresponded to unfiltered accelerometer output related to the heading of the magnetometer. Figure 8 shows the results of the accelerometer raw pitch and roll outputs.

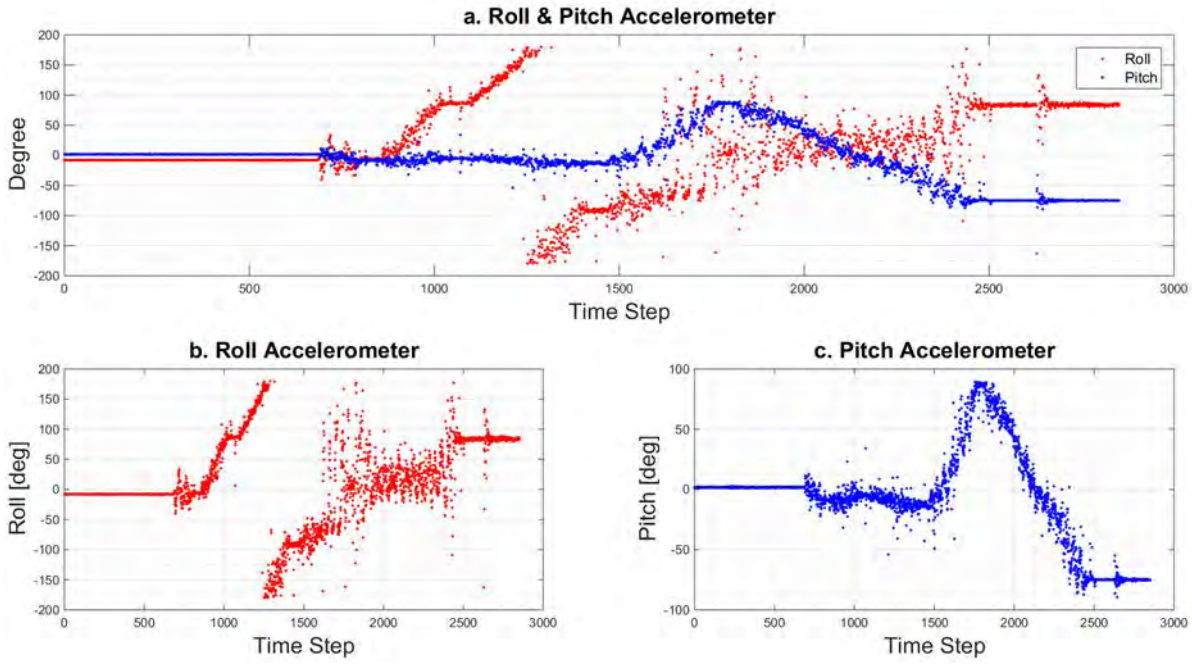


Figure 8 Accelerometer's Raw Pitch and Roll Output Results.

A rotate and translate pattern was programmed into the robotic arm. This included a non-translational 0° - 90° - 270° roll, followed by a 170° roll with a 90° - 90° pitch during translation. Time delays of 1 s where placed between each

sub-pattern. During the translation phase, a smooth vibration was observed in the arm.

The data shows noisy output, especially during the translation (time steps 1500 and 2450). During steady-state conditions, the accelerometer shows data inside parameters, as specified on [7].

For gyroscope tests, a program called "*giroraw"* was created. Giroraw receives the angular rate of each gyroscope and makes a zero-bias compensation by calculating an offset over 1000 samples. It then verifies noise over the average offset and integrates the drift filtered angular rate to obtain an angle, at a rate of 140 Hz. The *Jaco* rotate and translate pattern was used for this test. The results are displayed on Figure 9.

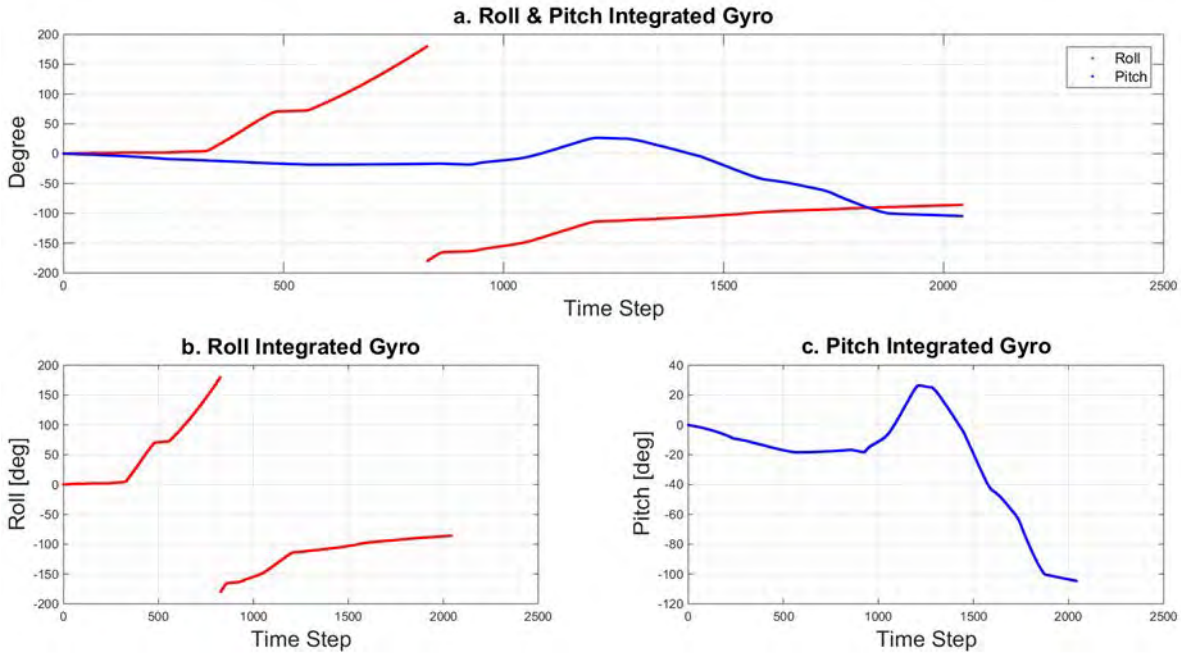


Figure 9 Gyroscope's Raw Pitch and Roll Output Results.

The results show a more stable and noise-free output during angular displacement, but the readings tend to drift over time, especially during steady-state conditions, as the zero-bias outputs becomes relevant and accumulates

over time. Zero-bias correction allowed to decrease the drift, but there are still errors that accumulate over time. Figure 10 shows a comparison between a compensated and non-compensated Zero-Bias gyro drift. The dramatic difference between them is due to an offset of 0.04 rad/s for the roll calculation. This is not trivial for steady-state conditions.

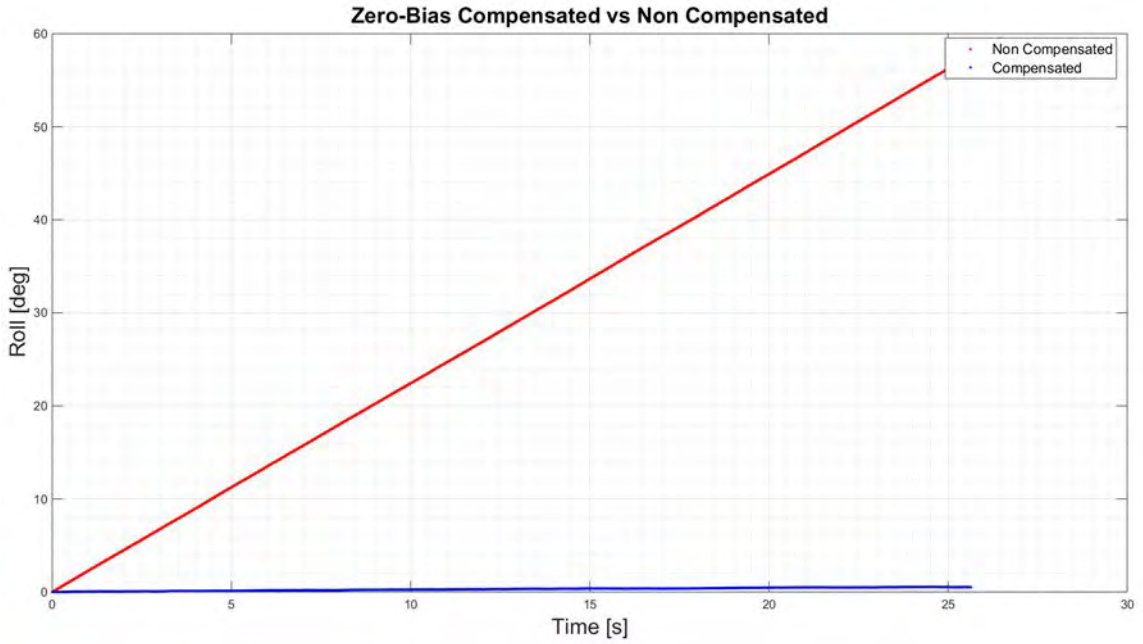


Figure 10 Gyroscope's Drift Comparison.

Bench tests conclusions:

1. The IMU was successfully bench tested and its performance was characterized. This helped understand the physical limits and performance characteristics if the IMU

2. Calibration of the magnetometer is required prior to operational use. The calibration must be done in an environment similar to the one that is being designed for (beach front) and with MOSARt fully operational. This calibration ideally must be done once and saved for future operations

3. The accelerometer raw data is noisy, but during steady state situations, it gives the desired pitch and roll reference. During angular movements, the gyroscope outputs useful roll and pitch data, but in steady state conditions it drifts over time

4. The gyroscope needs Zero-Bias compensation with respect to the three axes, for a reliable angle estimation

5. It is necessary to implement a fusion algorithm, based on gyroscope roll, pitch and yaw angle position. The accelerometer pitch and roll and the magnetometer (heading) will serve as reference data for the gyroscope

2. Adafruit Ultimate Global Positioning System (GPS)

Figure 11 Adafruit Ultimate GPS.

a. General Description

The GPS sensor, shown on Figure 11, is a 25.5 mm x 35 mm x 6.5 mm module built around the MTK3339 GPS chipset. Some of the capabilities of this chipset are [11]:

1. Satellites: 22 tracking, 66 searching

2. Frequency: L1, 1575.42 MHz

3. Built in passive patch antenna (size: 15 mm x 15 mm x 4 mm) and µFL connector for external antenna

4. Update rate: 1 to 10 Hz

5. Output: National Marine Electronic Association (NMEA) 0183 protocol, 9600 baud default

6. Multi Tone Active Interference Canceller (MTAIC) [12]: can cancel up to 12 independent interference caused by harmonic RF signals from another source (wireless communications, 3G/4G, etc.)
7. Acquisition sensitivity: -145 dBm, tracking sensitivity: -165 dBm
8. Voltage in range: 3.0-5.5 V DC
9. MTK3339 Operating current: 25 mA tracking, 20 mA current draw during navigation
10. Differential GPS (DGPS), Wide Area Augmentation System, (WAAS) European Geostationary Navigation Overlay Service (EGNOS) supported
11. Position accuracy: without aid 3.0 m (50% Circular Error Probability (CEP)), DGPS: 2.5 m (50% CEP)

b. Principle of Operation

The GPS receiver, with the aid of a built-in almanac, is able to receive and identify satellite signals within its area of operation. The receiver determines its position, relative to earth-center coordinates, through a satellite time-delay synchronization process. The velocity and heading of the satellites are obtained and used by means of trilateration technics, (spheres centered on each identified satellites with a radius equal to the distance (pseudo range) calculated with the time delay) to obtain a fixed position of the receiver relative to the earth-centered coordinate. Generally, four satellites are required to obtain a good fix. Increased accuracy can be obtained with the aid DGPS, which uses fixed ground stations in known positions to calculate differential corrections. As stated in [13] and [14], the accuracy of the GPS position is affected by a number of random and systematic errors, and are observed as signal propagation errors at the receiver. Techniques and mathematical models are used to mitigate these errors, but due mainly multipath errors and ionosphere delays, errors of several tenth of meters can be expected in normal operational conditions [14], [15].

c. Bench Tests

Basic operational tests were executed to determine an idea of the operational performance characteristics of the GPS. In all the tests, DGPS was achieved during the first seconds of reception, therefore the tests were conducted under optimal conditions.

The DUT was placed on a protoboard connected to an Arduino-DUE as a Pre-Processor (PP) and a notebook for data logging. The *Adafruit-GPS* library was used to establish control and data communication between the MTK3339 GPS chipset and the PP. The example program *due_parsing* was modified for the required output format. The GPS was configured to transmit at 9600 baud (recommended setting) and the selected NMEA sentences are Recommended Minimum (RM) and Global Positioning System Fix Data (GGA). The details of each NMEA sentences are exposed in [16]. The communication between the computer and the pre-processor was established at 115200 baud.

Tests were conducted in free and non-free multipath environments. A Matlab program *AXV_GPS.m* was created to display the variation in velocity, heading, altitude and position while the DUT was placed on the ground with no motion. Position error was calculated with respect a known geographic position. Note that the reference coordinates may not be exact, but they give an idea of the dispersion of the position. The results are presented in Figure 12, for a multi path environment (GPS antenna placed inside a corner wall) and in Figure 13, for an open area scenario.

Distance variations (top left) indicate the number of satellites for the largest and smallest errors. Calculations of distance and bearing between two closely spaced coordinates (less than a few kilometers) with the law of cosines resulted in unstable numerical solutions. Calculations by simple extrapolation of the latitude and longitude into a rectangular grid resulted in errors of more than 30° (this error is not distance dependent: it is latitude dependent). To overcome this problem, the bearing – distance polar plot (top right) and the distance

variation were calculated using the *Haversine law* [17] for distance (18) and azimuth (19). This formula takes into account the great circle through two points and it is suitable for small displacement (avoids numerical errors).

$$d = 2R_{earth}\sin^{-1}\left[\sqrt{\sin^2\left(\frac{\Delta_{Lat}}{2}\right)+\cos\left(Lat_1\right)\cos(Lat_2)\sin^2\left(\frac{\Delta_{Long}}{2}\right)}\right] \quad (18)$$

$$Az = \tan 2^{-1}\left[\cos(Lat_2)\sin(\Delta_{Long}),\cos(Lat_1)\sin(\mathrm{Lat}_2)-\sin(\mathrm{Lat}_1)\cos(\mathrm{Lat}_2)\cos(\Delta_{Long})\right] \quad (19)$$

In (18) and (19), *d* corresponds to the distance between the two coordinate points, R_{earth} to the Earth's radius, *Lat* / *Long* to latitude / longitude and Δ_{Lat} / Δ_{Long} to the difference between the two latitude and longitude. *Az* corresponds to the azimuth or true bearing between coordinate 1 and 2.

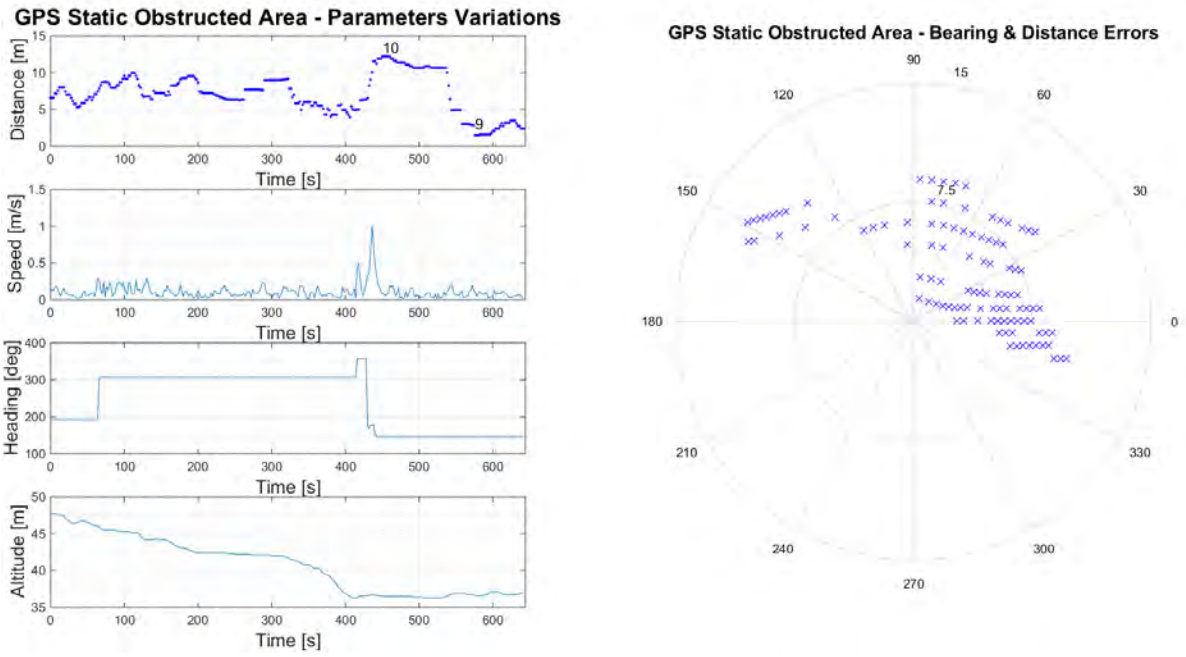


Figure 12 GPS Static – Multipath Test Results.

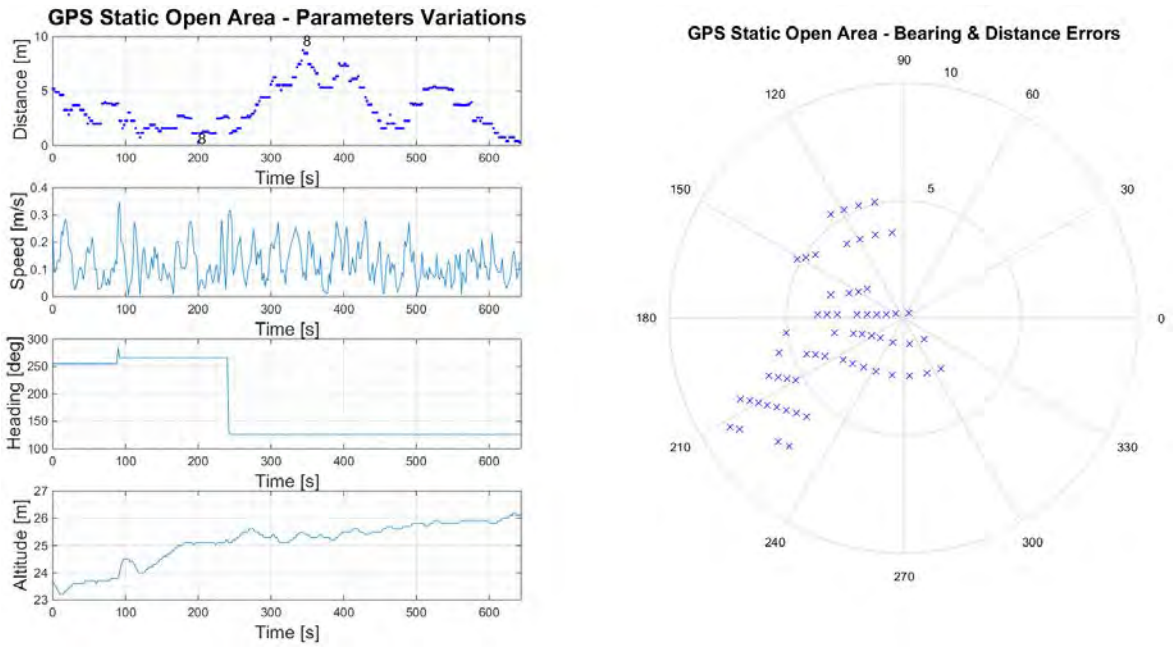


Figure 13 GPS Static – Open Area Test Results.

Figure 12 and Figure 13 show a GPS Fix dispersion of 12 meters in a multi-path environment (Figure 12) and 8 meters in an open area (Figure 13). These errors were obtained during DGPS operation, with 10 and 8 satellites being tracked. This position displacement affects the speed and heading calculation of the GPS platform and, in the case of Figure 12, it coincides also with a change of behavior of the altitude output, which indicates an error on the pseudo range calculation, probably due to multipath effects. The *AXV_GPS.m* program also calculated the percentage of fixed points inside a 2 [m] error ellipse. For the open area it gave 32% and for the multipath environment 4%.

Dynamic tests were conducted in a free and non-free multipath environment. Basically, a “walk through the park” was done. Results are displayed in Figure 14, using a free online Latitude and Longitude track viewer [18]. As shown in Figure 14 (a), (b) and (c), GPS fixes vary throughout the day. For all data sets it was observed the GPS receiver maintained signal with 7 to 8 satellites in DGPS mode. Buildings, trees were determined to be a source of degraded signal.

Figure 14 GPS Dynamic Test Results.

Some preliminary conclusions:

1. The number of satellites tracked in DGPS mode does not correlate to a fix within required operational position, heading, speed and altitude specifications. External factors (multipath, ionosphere, etc.) contribute to the errors

2. Poor performance was observed during operation in relative open areas. This is attributed to the low gain performance of the passive patch antenna. An external active antenna must be considered for a better multipath cancelation

3. The Haversine law proved effective in bearing and distance calculations between sub-meter separated coordinates

4. Update delay: every 1 s, one of the selected NMEA messages is transmitted to the pre-processor, hence useful data is available to the receptor computer every 2 s. This could cause a significant delay time in the receptor computer if it is not handled properly.

5. Transmission format: There are drawbacks to the use of ASCII characters for long data string transmissions. First, data packages do not maintain a fixed length, which obligates the receiver to wait for additional bytes. Secondly, this causes a 20 ms delay during transmission,

3. SparkFun Pressure Sensor

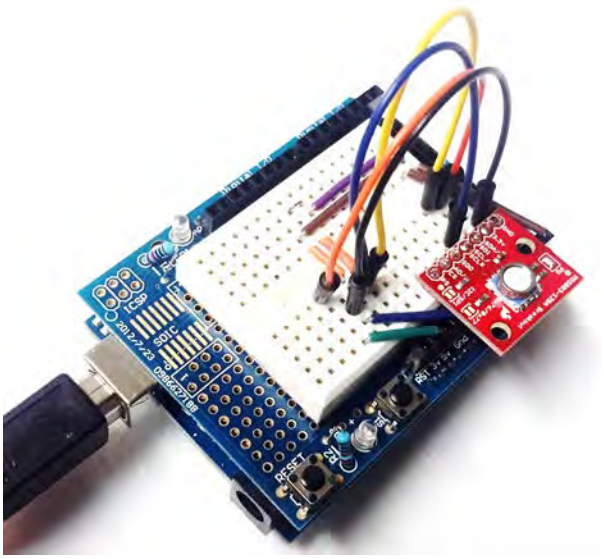


Figure 15 MS5803-14BA Pressure Sensor over a SparkFun Circuit Board (Red).

a. General Description

Figure 15 shows the MS5803-14BA pressure sensor (top left: white-top cylinder) mounted over a 18.5 mm x 20.5 mm x 2.0 mm circuit module manufactured by SparkFun (red color). Some of the main characteristics of this board are [19]:

1. Pressure range: from 0 to 1.4 MPa for full accuracy. Can withstand up to 3 MPa
2. Resolution up to 20 Pa
3. Temperature reading range: from -40°C to 85°C, with a resolution in the order of 0.01°C
4. Pressure sensor: MEMS high linear piezo-resistive pressure sensor
5. Support I2C protocol for data transfer

Comparison of the external pressure with the existing pressure in a small sealed reference chamber determines absolute pressure. To maintain linearity, temperature is taken into account. The sensor has a built-in first and second order temperature compensation for this purpose.

The absolute pressure will be used to calculate the depth by means of the following formula:

$$d = \frac{(P_{abs} - P_{sea\_Level})}{\rho g} \tag{20}$$

P_{abs} refers to the measured pressure (underwater) and P_{sea_Level} to the pressure measured at the surface. In the denominator, ρ accounts for density of water (or seawater) and *g* for gravity.

b. Bench Tests

The DUT was waterproofed with an acrylic conformal coating to seal the circuit boards. The electronics box was filled with standard wax and liquid electrical tape was used for final protection. A box was designed for UW sensor tests. The box was modeled in SolidWorks and 3D printed by [2]. Figure 16 shows the general layout of the box prior to waterproofing. The box was attached to a metal rod with marks at 25, 50 75 and 100 cm.

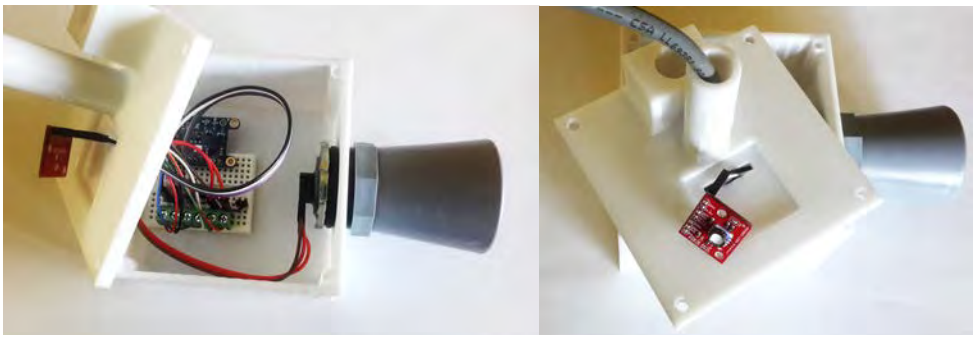


Figure 16 Pressure, Sonic and IMU Sensor in UW Box Assembly.

The program *AXV_pressureUW07JUN* was created, using the SparkFun's library for temperature and pressure raw readings for calculating depth. The output was saved and analyzed in Matlab by the program *AXV_pressure.m*. The results are displayed in Figure 17.

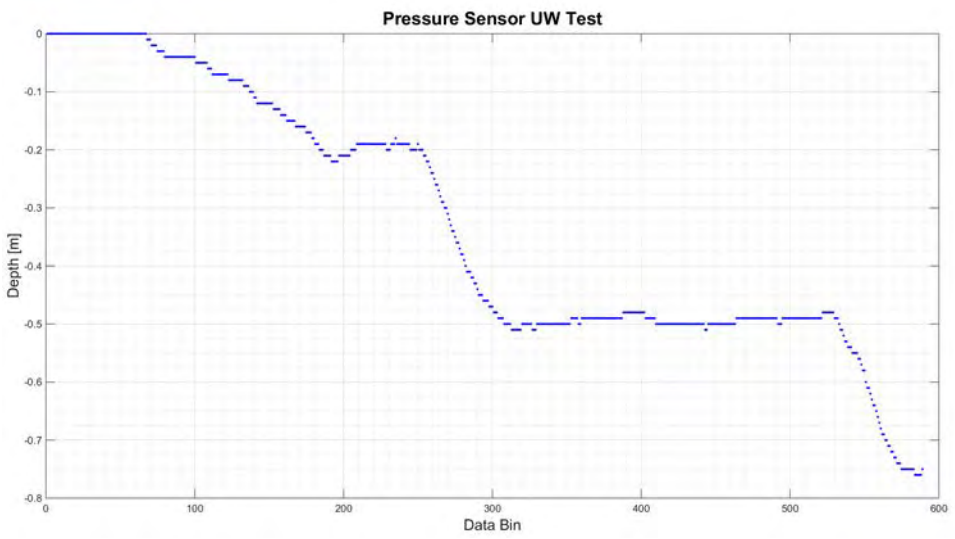


Figure 17 UW Pressure Sensor Test Results of Depth Calculation.

Figure 17 shows depth readings with a resolution of 1 cm. The measurement were consistent with handmade measurements, validating equation (20).

4. HB100 Doppler Speed Sensor

Figure 18 HB100 Patch Antenna.

a. General Description.

This is a miniature (46 x 40 x 7 mm) bi-static X band Doppler transceiver module, as shown on Figure 18. It can be configured to function pulse based or continuous wave (CW) operation. None of the previous configurations will achieve distance measurements, as this device is primary used for vehicle speed measurements. Some of its main characteristics [20]:

1. Frequency: 10.525 GHz
2. Radiated power: 15 dBm
3. Antenna: patched antenna (bi-static)
4. Azimuth beam width (3 dB): 80°
5. Elevation beam width (3 dB): 40°
6. Supply voltage: 5 V
7. Current consumption: 30 mA
8. Output: train pulse representing absolute Doppler shift

b. Principle of Operation.

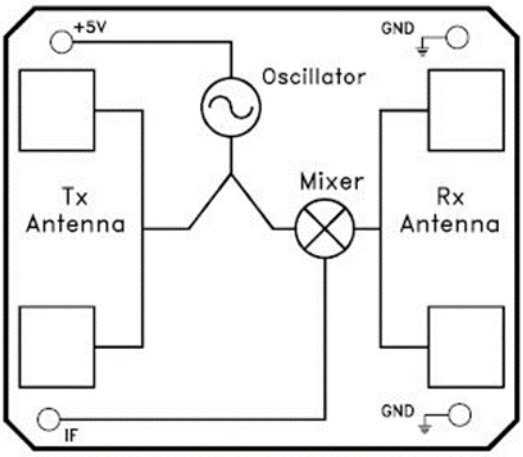


Figure 19 HB100 Block Diagram.

Source: [20]: Agilsense, *HB100 10.525 GHz Microwave Motion Sensor Module Version 1.02*. Jurong: ST Electronics.

Figure 19 shows a simplified block diagram of the HB100 sensor. The voltage of the transmitted signal is considered to have the form:

$$v_{tx}(t) = A_{tx}\sin(2\pi f_{tx}\,\mathrm{t} + \phi_{\mathrm{o}}) \tag{21}$$

A_{tx} is the amplitude of the transmitted signal, f_{tx} is the frequency of the transmitted signal and ϕ_{o} is the phase at t = 0. For this explanation, ϕ_{o} will be set to zero. If the signal detects a target, the received signal will present a delay equal to:

$$\tau = \frac{2R}{c} = 2\left(\frac{R_o \pm \dot{R}t}{c}\right) \tag{22}$$

2R represents the two-way travelling of the electromagnetic wave, R_o is the initial range, $\dot{R}$ is the velocity of the target (assumed constant) and *c* is the speed of light (note that as the HB100 is a CW transceiver, the time delay or range cannot be measured, as there is no time reference). Considering this time delay, the received signal will have the form:

$$v_{rx}(t) = A_{rx}\sin\left[2\pi\,\mathrm{f}_{\mathrm{tx}}\left(t - \frac{2(R_o \pm \dot{R}t)}{c}\right)\right] \tag{23}$$

Grouping the values, and considering the case of an incoming target (negative velocity), the received voltage signal becomes:

$$v_{rx}(t) = A_{rx}\sin\left[2\pi f_{tx}t\left(1 + \frac{2\dot{R}}{c}\right) - \frac{4\pi\,\mathrm{f}_{\mathrm{tx}}\,R_o}{c}\right] \tag{24}$$

Note that the second term of the sine function of (24) is constant. Referring back to Figure 19, the received signal is mixed with a sample of the transmitted signal (21), then, following the trigonometric identity

$$\sin(x)\sin(\mathrm{y}) = \frac{1}{2}\left[\cos(x-y) - \cos(x+y)\right] \tag{25}$$

The difference signal is extracted, so the output of the sensor will present the form:

$$v_{out}(t) = A_{out}\cos\left[2\pi f_{tx}\frac{2\dot{R}}{c}t - \frac{4\pi \mathrm{f}_{\mathrm{tx}} R_o}{c}\right] = A_{out}\cos\left[2\pi\frac{2\dot{R}}{\lambda}t - \frac{4\pi \mathrm{f}_{\mathrm{tx}} R_o}{c}\right] \quad (26)$$

The above expression shows that if a target is not moving, there will be a constant output. To relate (26) to the Doppler frequency measurement, it is necessary to analyze the Doppler frequency shift of a moving target return. The first step is to account for the total number of wavelengths that fit in the two-way path and consider that each wavelength corresponds to a phase shift of 2π, then the total phase shift that the signal will suffer in the two-way path will be

$$\phi = 2\pi\frac{2R}{\lambda} = \frac{4\pi R}{\lambda} \quad (27)$$

If *R* presents time dependence (hence the target is moving) with respect to the source (it has to present a radial velocity), the rate of change of the phase can be extracted by differencing (27) with respect to time:

$$\omega_d = \frac{d\phi}{dt} = \frac{4\pi\dot{R}}{\lambda} \quad (28)$$

where λ accounts for wavelength. This phase rate of change is caused by the Doppler frequency shift f_d. Since $\omega = 2\pi f$, then (28) becomes, after solving for f_d:

$$fd = \frac{2\dot{R}}{\lambda} \quad (29)$$

Relating (29) with (26), a final expression can be obtained

$$v_{out}(t) = A_{out}\cos\left[2\pi f_d t - \frac{4\pi \mathrm{f}_{\mathrm{tx}} R_o}{c}\right] \quad (30)$$

Equation (30) indicates that if the target is moving, the output of the receiver will present a frequency equal to f_d, which is proportional to the radial velocity of the target and the transmitted frequency (or wavelength).

Finally, if the target is not moving directly into the source (with an angle θ off-axis), only the radial component of the velocity will be considered in (29). To account for this, (29) has to become

$$fd = \frac{2\dot{R}}{\lambda}\cos\theta \tag{31}$$

Expression (31) fits the equation given in [21].

c. Bench Tests.

The HB100 sensor is used to measure the speed over ground. For speed readings, a microcontroller is used to measure the frequency of the output pulses of the HB100 (which corresponds to the Doppler frequency) and apply equation (31) to calculate the speed.

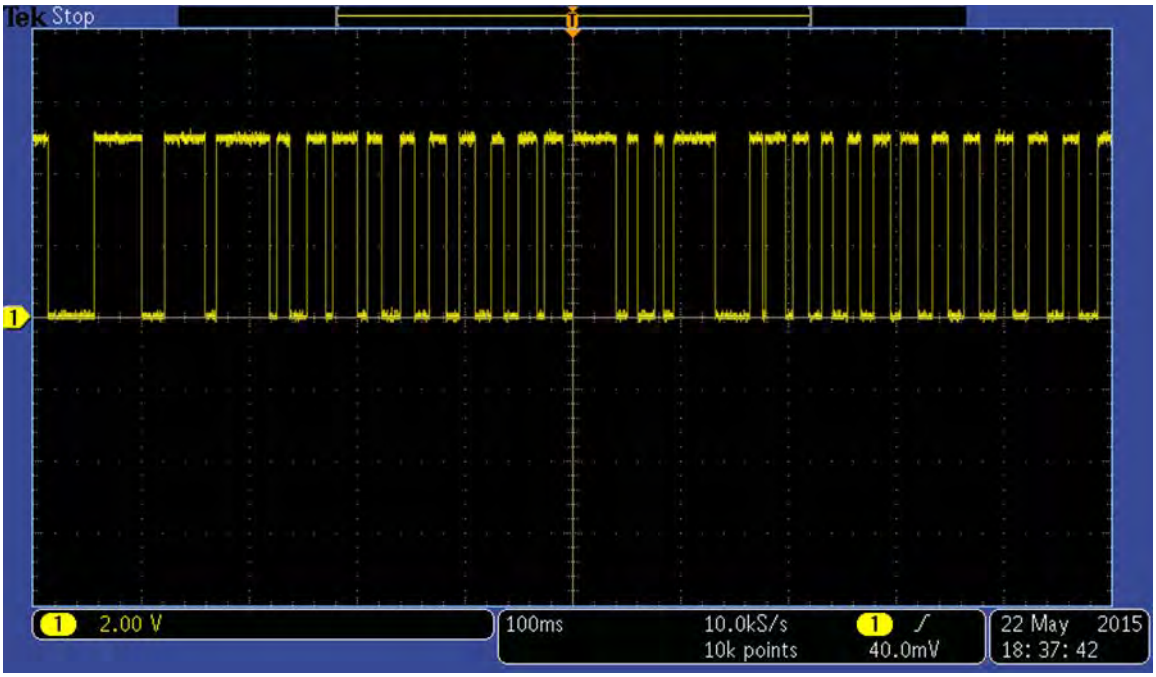


Figure 20 HB100 Raw Reading Example.

The first step was to analyze the raw output readings on an oscilloscope (see Figure 20), obtaining the following observations:

1. The device is very sensitive, thus small vibrations produces valid readings

2. All pulses presents the same amplitude (due the configuration of the amplifier)

3. The HB100 does not give a reference to determine if the Doppler frequency is positive (incoming target) or negative (increasing distance)

4. As shown on Figure 21, the antenna presents a low directionality. This situation, plus multipath readings obtained in a confined room produced unwanted readings from the sides and back of the HB100

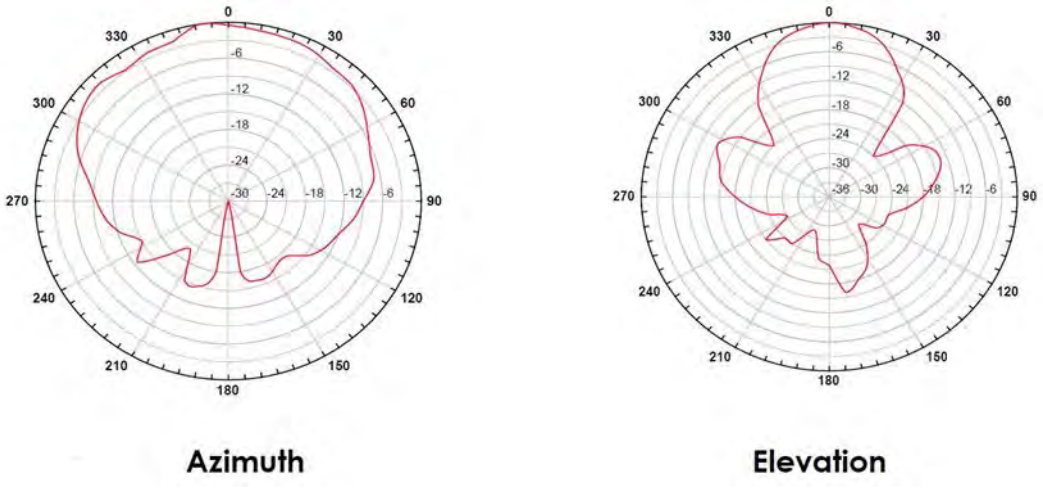


Figure 21 HB100 Antenna Beam Pattern.

Source: [20]: Agilsense, *HB100 10.525 GHz Microwave Motion Sensor Module Version 1.02*. Jurong: ST Electronics.

The second step was to observe the accuracy of the readings with a microprocessor. As MOSARt should produce Doppler frequencies around 30 Hz, it is best to measure the elapsed time during each cycle of the input frequency by means of an internal counter. This functionality was tested using an external function generator. The Teensy 3.1 microprocessor was selected for this task, since it can achieve an accuracy of better than 0.01 [Hz] for the frequency measurement.

To simulate displacement, the *KINOVA Jaco robotic arm* was used, by placing the sensor normal to a concrete wall (see Figure 22). A pre-established pattern was programed and different velocities were programed, taking care to minimize off axis movements.

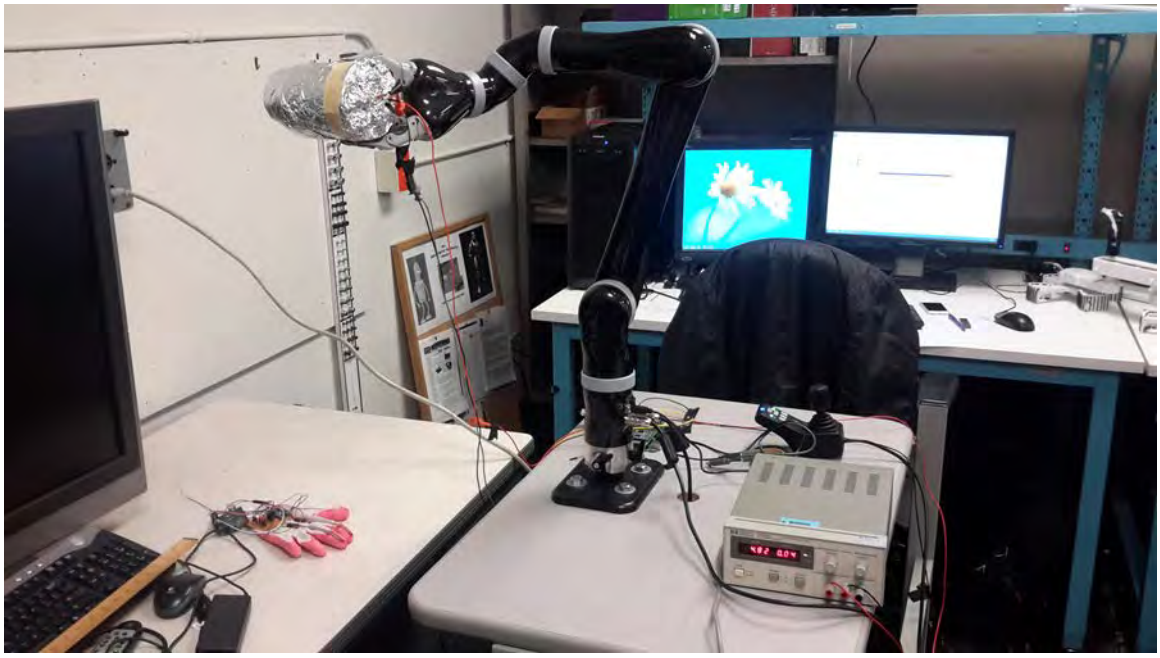


Figure 22 HB100 Bench Test Setup with Jaco Arm.

The program *AXV_displ.c* was made using a trapezoid integration to calculate the final displacement. To reduce side lobe detection, the HB100 was enclosed on a 3D printed cylinder and its sides and back coated with aluminum foil. Two runs are displayed in Figure 23. The final displacement was calculated within 5% of the measured distance (less distance was measured). The constant velocity of the arm can be observed in the slope of the two data points, which coincide with the real velocity set in the arm. A reduction in velocity can be noticed in the final segment of the 20 m/s run.

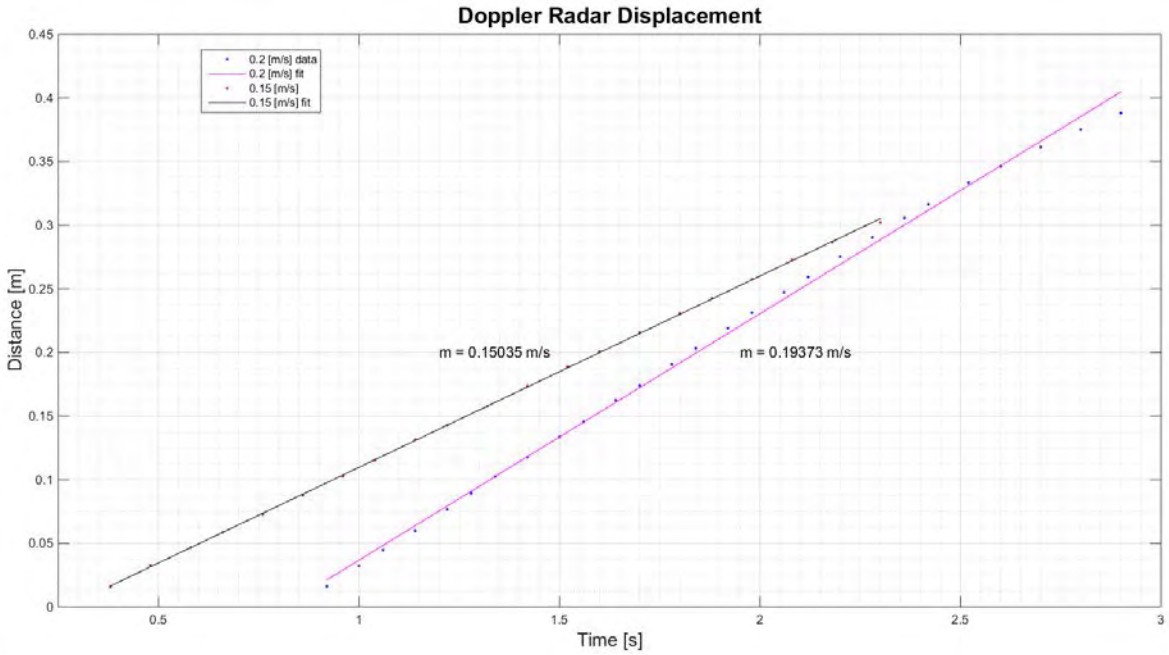


Figure 23 HB100 Displacement Calculation with Jaco Arm.

Some preliminary conclusions of these tests:

1. Velocity measurement by means of Doppler theory gives an accurate displacement calculation, but the implementation of this technique will be affected by the vibrations and irregularity of the terrain

2. Side lobe cancelation must be improved, as the aluminum cover does not absorbs unwanted side lobe signals that pass through the front cover of the cylinder

3. The minimum detectable speed is 10 cm/s. This limitation, plus the fact that the radar did not maintained a normal incidence during the displacement of the arm could explain the lower displacement measured (within 5%)

4. Vibration sensitivity can be used to identify the type of operational terrain

5. HRXL-MaxSonar-WR MB7360

Figure 24 MAXSONAR Sonic Sensor with Horn.

a. General Description

This family of sonic transducers compromises a rugged waterproof sensor (see Figure 24) and a flexible processing and communication system. Some of its main characteristics are [22]:

1. Frequency of operation: 42 kHz
2. Modulation: train of pulses, Pulse Width ~118 ms
3. Noise tolerance: better than 106 dB for a 9 cm diameter dowel (target) at 3.7 m
4. Resolution: 1 mm (pulse width reading), 5 mm (analog reading)
5. Waterproof standard: IP-67 (supports 1 m immersion for 30 minutes without additional protection)
6. Voltage of operation: 3.3 – 5 V
7. Current draw: 2.3 mA (3.3 V), 3.1 mA (5 V).
8. Built in clutter rejection, noise filtering, voltage and temperature compensation (for temperature it requires external temperature readings)
9. Analog or PWM output
10. Free run, single trigger or multiple sensor operation modes
11. Update frequency (single sensor): 7 Hz

b. Principle of Operation.

These ultrasonic transducers transmit a modulated (pulsed) acoustic pressure wave. The interaction of the acoustic wave with an object will, in general, reflect the wave back to the transducer, which will calculate the distance of the object by means of the time of flight of the transmitted signal, using the same relation described on the left hand side of equation (22), but this time c will be equal to [23]

$$c = \sqrt{\frac{\gamma RT}{M}} \tag{32}$$

In the last equation, *R* represent the gas constant, γ the heat capacity ratio (C_p/C_v), *T* the temperature in degrees Kelvin and *M* is the molar mass of the gas. As temperature is an unknown parameter, the MB7360 sonic detectors has a built-in temperature sensor. With this, we can compensate for the 0.6 [m/s] variation per degree Celsius in the speed of sound [22].

To create the desired ultrasonic pressure wave, movement of a surface is needed [24]. This is achieved by means of a piezoelectric transducer operated in the motor mode. The transducer converts electrical energy into mechanical energy. When the incoming echo signal is received, the piezoelectric ceramic will flex, generating an electrical signal that will be compared to a time adaptive threshold. The MB7360 series incorporate filtering techniques to reduce false alarms. After the transmission, the piezoelectric device rings for a finite amount of time. This causes degradation in the distance measurement performance for targets detected in the 30-50 cm range. The acoustic effects in the near field will be affected by acoustic phase cancelation of the echo, resulting in inaccuracies up to 5 mm in this operational range [22]. Below 30 cm, only presence of a target is signaled.

Beam width is inversely related to the frequency and the antenna size. Wide beam widths can be managed by a feed horn implementation as shown in Figure 24.

As all the sensors presents the same operational frequency, mutual-interfering between sensors must be taken into consideration when implementing an antenna array solution. The MB7360 series allows one to interconnect in sequence the transmission commands of these sensors, avoiding interference, but with the price of a slower update rate.

c. Bench Tests

Figure 25 shows the test set up for a single sensor. The sensor's analog output was chosen, as the digital output was considered too slow.

Basic distance measurements were compared, obtaining readings within ± 1 cm of accuracy. As stated during the principle of operation description, the angle of incidence is a critical factor for detection, obtaining irregular readings (no detection) with angles greater than 30°.

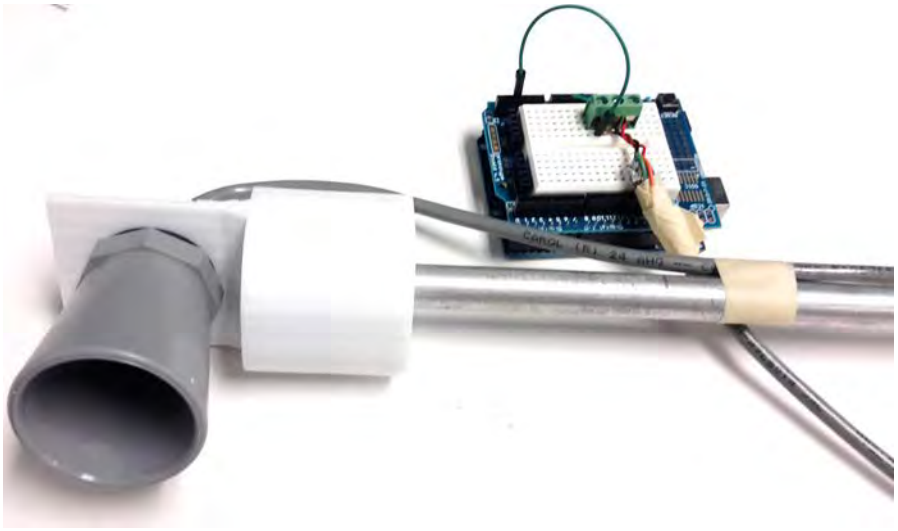


Figure 25 Single MAXSONAR Test Set Up.

For the dynamic test, the *KINOVA Jaco robotic arm* was used to move the sonar in a repeated pattern with two sets of constant velocities (15 m/s and 20 m/s). The Arduino UNO was programmed to sample at twice the refresh frequency of MAXSONAR. Figure 26 b and c show a zoom-in of the data during displacement at 15 m/s and 20 m/s. A linear fit was included to compare the

calculated slope with the real velocity of the arm. Motionless readings showed a variation of ± 1 cm.

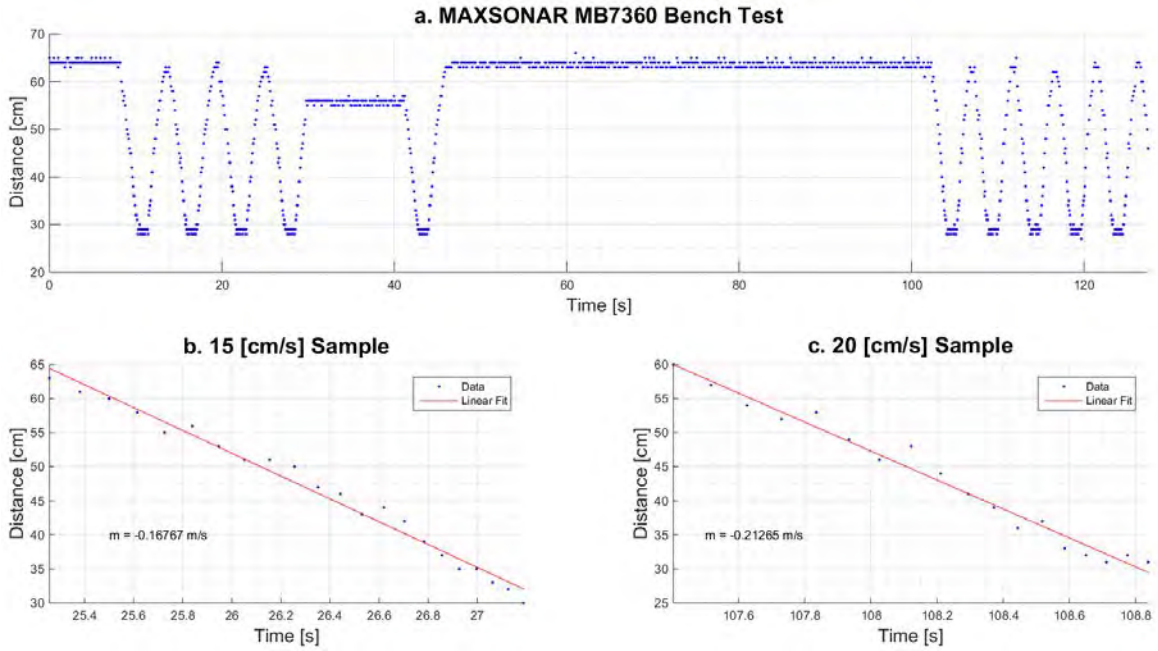


Figure 26 MAXSONAR HB7360 Data Output.

Tests were conducted using more than two MAXSONAR sensors running in free mode (not synchronized) transmitting from the same origin. For angles of separation less than 45°, interference was observed between the sensors, showing a false target at 30 cm when no target was present.

In summary, the sensors performed as expected. However, mutual interference proved problematic.

6. DST800 TRIDUCER

Figure 27 DST800 TRIDUCER Smart Sensor

a. General Description

The TRIDUCER, Figure 27, is a multisensor device that provides: depth, speed and temperature capabilities in one device. It is a “smart sensor,” in the sense that the output is processed as digital information in the NMEA 0183 data format. Some characteristics include:

1. Echo sounder frequency: 235 KHz
2. Transducer beam: wide fan-shaped
3. Depth range: 0.5 – 70 m
4. Log: paddle wheel type, with speed signal processing for speeds less than 5 knots
5. Data uprate rate: 1 per second
6. Data output: NMEA 0183 over a RS422 hardware protocol, 4800 bauds
7. NMEA sentences: $SDDBT, $DDPT (depth); $VWVHW (speed), $VWVLW (distance), $YXMTW (water Temperature)
8. Supply voltage: 10 to 25 DC

9. Supply current: 40 mA

b. Principle of Operation

The echo sounder is based on the same principle described for the HRXL MaxSonar transducers. For log measurements, a Hall sensor detects a spinning of a propeller that rotates at a speed proportional to the speed over ground. Data is collected via the RS422 protocol (see Figure 28).

c. Bench Tests

To bench test, the TRIDUCER was placed in a bucket of water and its signal output (Tx+ and Tx-) analyzed on an oscilloscope. Figure 28 presents the RS422 balanced signal output.

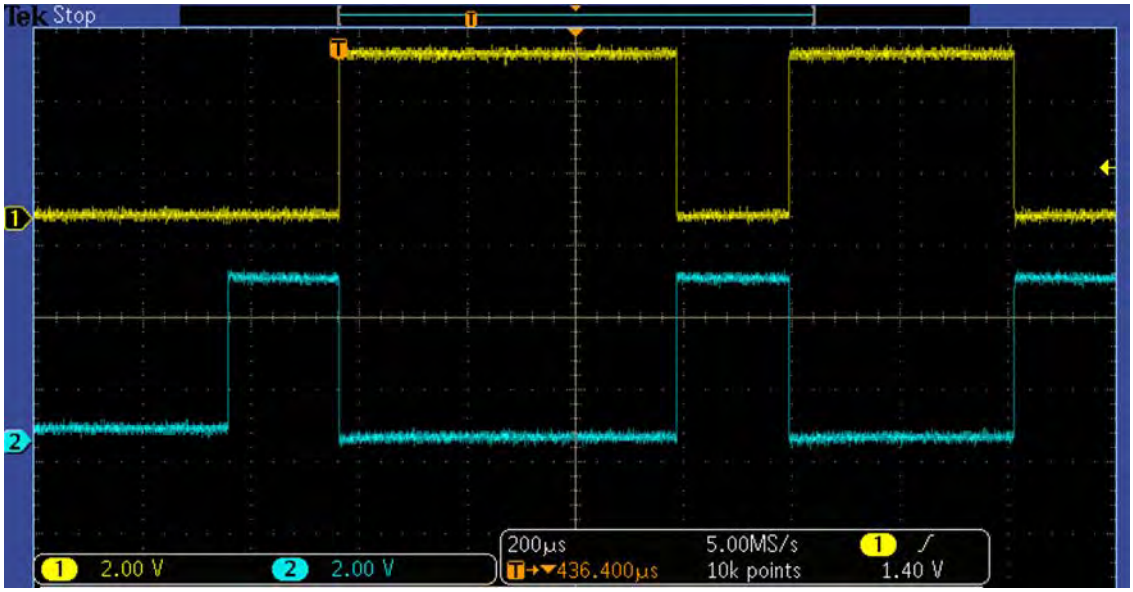


Figure 28 DST800 RS422 Output.

To connect the TRIDUCER to an Arduino processor, a protocol converter RS422 to RS232 was needed. The DST800 sent depth, temperature and speed data package every 2 seconds in NMEA format. Speed information is sent once the speed is first sensed (simulated by manually rotating the wheel pad). The device presents a delay of around 4 s, probably due to internal processing and

filtering functions. To apply this device on MOSARt, a parsing library will have to be developed and implemented in a pre-processor, in order to have the data ready on the correct format, avoiding the 2 s refresh update time.

7. Geetech Infrared Proximity Switch Module

a. General Description

This is a low cost Infra-Red (IR) proximity switch. It does not output distance, but a flag is set when the detector senses a distance higher than a used-defined threshold. Some of its characteristics are:

1. Power Supply: 5 V DC
2. Supply current DC < 25 mA
3. Maximum load current 100 mA (Open-collector NPN pull-down output)
4. Response time < 2 ms
5. Pointing angle: ≤ 15 °, effective from 3-80 cm Adjustable

b. Principle of Operation

This photoelectric sensor is composed of a transmitter and receiver circuit. The transmitter outputs a pulse modulated IR signal. The receiver detects this modulated signal and compares it with a pre-established threshold (configured with a built in potentiometer). The threshold defines a distance, hence when the signal is outside this threshold, it outputs a logic 1.

c. Bench Tests

The IR sensors are used to protect the vehicle from “pothole” dangers. Sensors are placed near each wheel, and face forward at a maximum incidence angle of 15°. A four-sensor array was used (see Figure 29) to simulate one pair of sensors for forward and aft displacement. Each pair was powered independently. The four outputs were integrated with a 7432 logic OR gate. Pull-up resistors were used to ensure a defined logic 1 or 0 (especially when one of the sensor pairs was powered off).

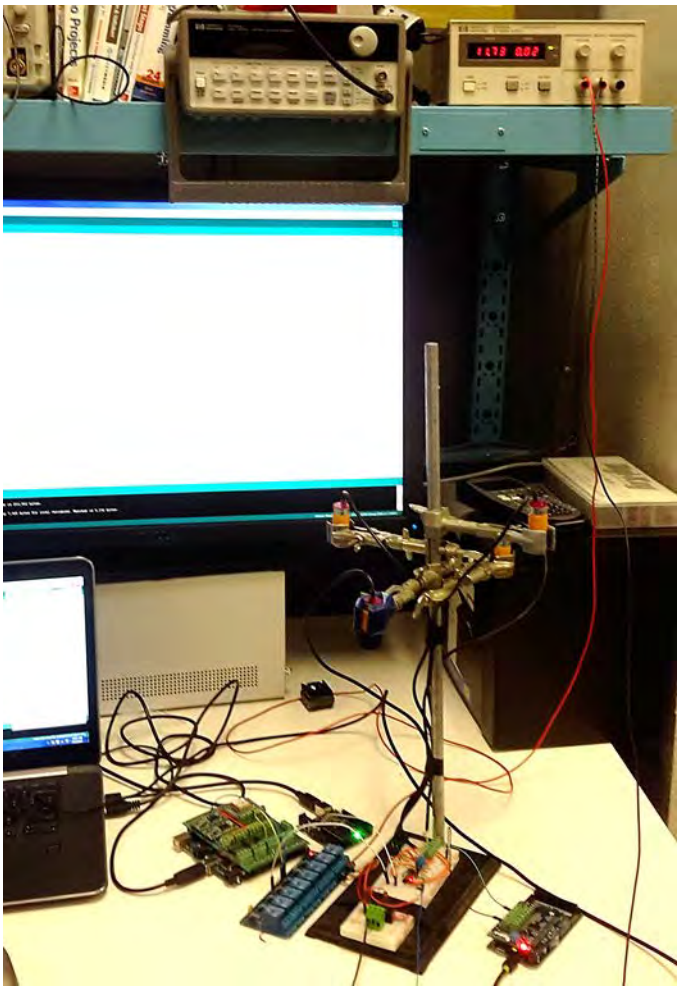


Figure 29 IR Switch Array During Laboratory Test.

Some preliminary conclusions of the test:

1. No mutual interference was observed
2. The threshold setup is sensitive to the voltage variations, so a steady supply must be guaranteed

8. Adafruit 16-Channel 12-Bit PWM/Servo -I²C Interface

Pulse Width Modulated (PWM) control was invoked via an external 12-bit external PWM interface. This was done to free up microcontroller resources.

This device comes in a shield compatible with the Arduino family and relies on the I2C protocol for command transfers from the Arduino. It has 16 PWM outputs. PWM signals are a common way to interface with an Electronic

Speed Control (ESC) device. The ESC interprets the average voltage achieved with the PWM signal, which is used as a command for managing the power (speed and direction) of the motor.

9. Motor Controllers / ESC

An electronic speed control device was used to manage five motors on the vehicle. This includes three sea thrusters and two land motors. Tail deployment is achieved by two servos that do not require a controller. Six of seven actuators use the IMU as a primary feedback mechanism for the closed loop control. Thruster vertical thruster uses depth information for feedback control.

Each sea thruster come with a built in ESC. PWM is used for communication to the processor. For the land and tail deployment motors, the RoboteQ SDC2120S dual/ single Motor Controller has been selected, due to its size and power management capabilities. PWM has also been selected for communicating to the PID processor. Each motor controller can hold a continuous current of 40 [A] (single channel operation). More current can be achieved by proper cooling.

10. Computing

Three families of microprocessors/microcomputers are used for the construction of this prototype: The Arduino, the Teensy (microprocessors) and the Raspberry-Pi (microcomputer). Each microprocessor has different characteristics. Table 1 summaries some of the main characteristic of the boards used in this study:

Table 1 Microprocessors and Microcomputer Main Characteristics.

	Arduino MEGA	**Teensy 3.1**	**Raspberry-Pi 2B**
Processor/ microcontroller	ATmega1280	MK20DX256VLH7 Cortex-M4	quad-core ARM Cortex-A7
Speed	16 [MHz]	96 [MHz]	900[MHz]
Operational System	none	none	Raspbian
Voltage of operation	5 [V]	3.1 [V]	3.1 [V]
Digital IO	54	34	21
Analog inputs	16	21	None
Analog outputs	None	2	None
Communications	Serial, I2C, SPI, USB	Serial, I2C, SPI, USB, CAN bus	Serial, I2C, USB, etc.
Language	C	C	C, Python, etc.

In general, the microprocessors are used as preprocessors (PP) to process sensor outputs, transforming the data (which can come in form of an analog signal, train of pulse, raw data, etc.) to the required format and units. These tasks include filtering and data fusion (if necessary). For programming, the C language has been selected.

The microcomputer (Raspberry-Pi) will be used as the Main Processor (MP). It will receive pre-processed data from the microcontrollers and will integrate the information to perform global tasks such as path-planning, sensors and actuators commands, external communications, etc. Python3 is used as the primary programming language.

THIS PAGE INTENTIONALLY LEFT BLANK

III. INTEGRATION

The integration of the sensors and devices requires different communications protocols. This includes various timing, voltage and post processing demands. Figure 30 shows a block diagram of the preprocessors and how they interact with the main processor. All of these devices are installed inside the waterproof cylinder.

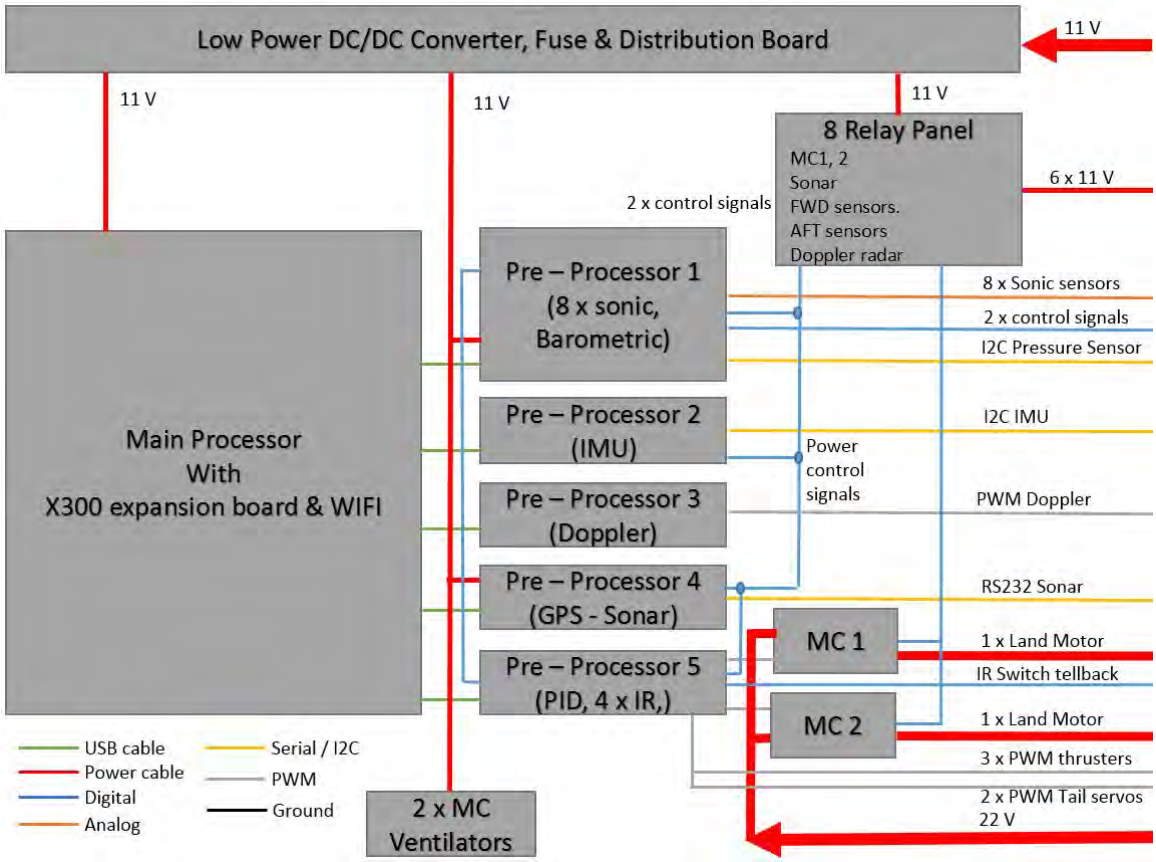


Figure 30 MOSARt Interior-Cylinder Block Diagram.

Sensors located outside the cylinder are protected with a special waterproof coating, see Figure 31.

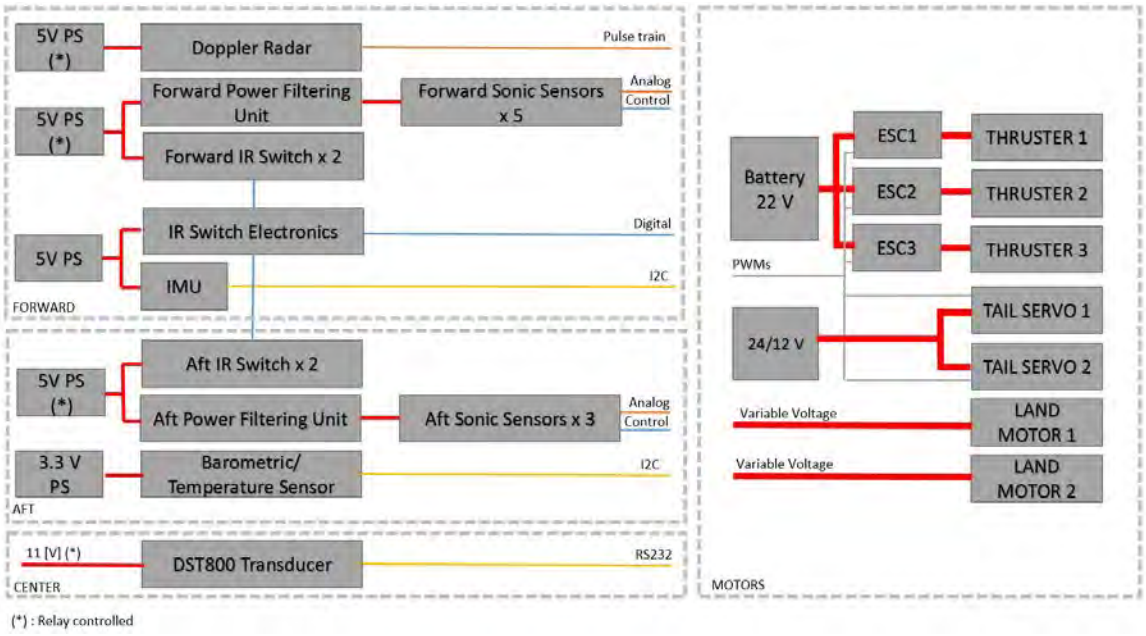


Figure 31 MOSARt Outside-Cylinder Block Diagram.

We address systems integration in two sections as follows:

A. HARDWARE INTEGRATION.

1. Exterior Design.

Figure 32 shows the general sensor layout. In the forward section, a MaxSonar sensor array is installed at a position that avoids interference with the Whegs. At the center point of the array, the IMU is installed. This position avoids parallax corrections with respect the forward MaxSonar sensor array. Each Wheg position uses a forward (aft for the rear Whegs) looking IR Switch that points at a 10˚ incidence angle. Forward and aft IR switches output signals converge into a logic signal integration assembly. Both IR Switch arrays (forward and aft) are powered in parallel to their corresponding MaxSonar array. Doppler radar is placed inside a tube facing at a 45˚ incidence angle with respect to the ground. In order to reduce the radar's side lobe interference, the interior of the tube is coated with radiation absorbing sheet, tuned for 9 to 10 [GHz] (see Figure 33). Above the aft MaxSonar array, the barometric/ temperature sensor is placed.

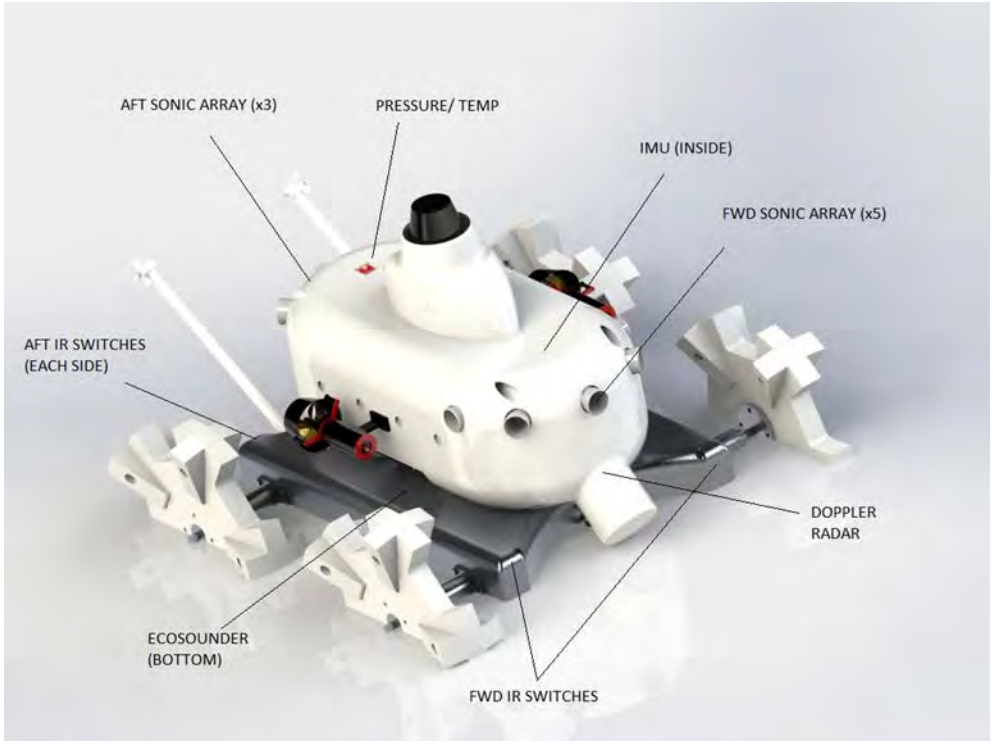


Figure 32 MOSARt General Exterior View and Sensor Layout

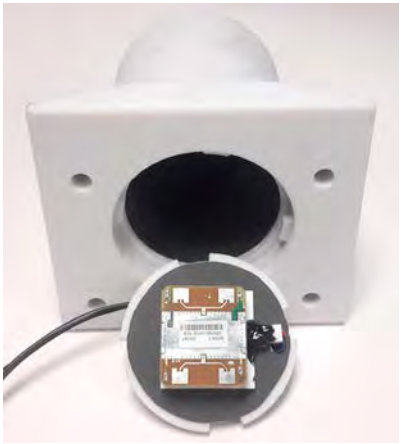


Figure 33 Doppler Radar Radome, with Interior Radiation Absorbing Material.

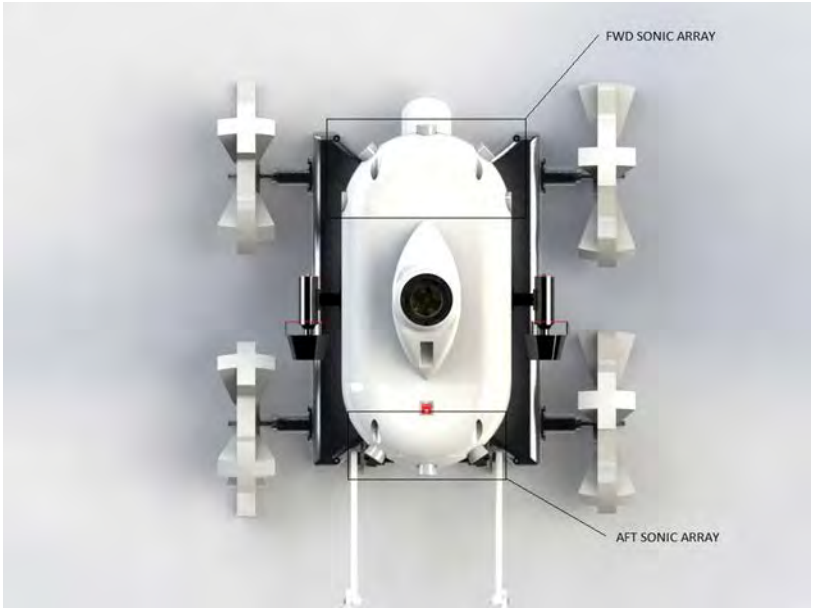


Figure 34 MaxSonar Sensor Array Layout.

Figure 34. Both have independent power supplies. The Forward array has 5 sonic sensors, each with an angular separation of 45˚. The Aft array has three sensors, with a 30˚ angle of separation.. To avoid interference, the transmit control line has been daisy chained, as shown in Figure 35. The PWM output is used as a trigger input for the next MaxSonar sensor.

The forward triggering sequence as configured as follows: center (0°) – center right (45°) – center left (-45°) – right (90°) – left (-90°) sensor. This configuration allows each device to range only after the previous sensor has finished [25]. The trigger output of the last sensor is connected through a 1 KΩ resistor to the trigger input of the first sensor. This allows a continuous loop operation. The same configuration is applied to the aft array. To initialize and maintain the continuous loop, the microcontroller's output pin is pulled high for 20 µs, then the pin is returned to a high impedance state. When the loop is complete, the trigger signal output from the last sensor will trigger the first sensor.

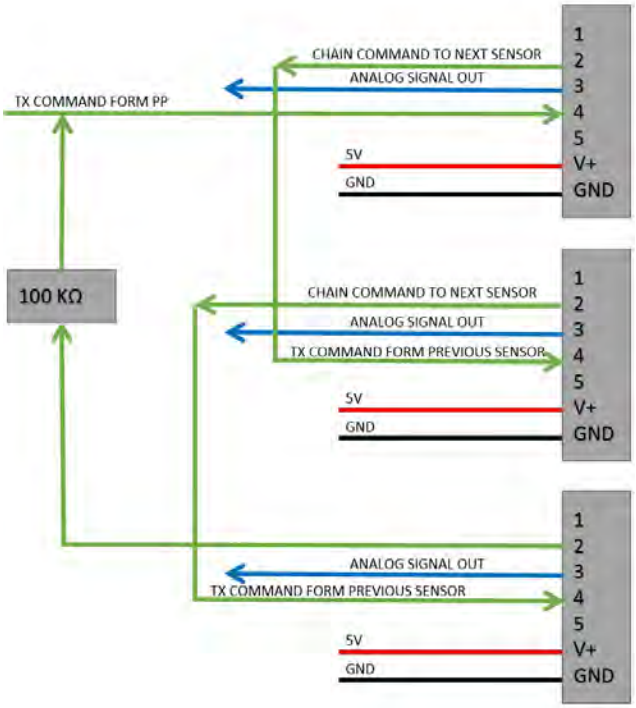


Figure 35 MaxSonar Multiple Sensor Connection (Three Sensor Example)

For signal distribution and power conversion, the following electronic board was designed and constructed (refer to Figure 36 and Figure 37):

1. The Forward Power Converter and Distribution Unit (FPCDU): this unit executes the power conversion and distribution to all the forward sensors (IMU, Doppler, IR Switch, Forward Array), as shown in figure 35. It also manages the incoming and output signals of the sensors. The power to the MaxSonar array comprises a low pass filtering unit for each sensor. The FPCDU holds the IMU unit as a mezzanine

2. The Aft Power Converter and Distribution Unit (APCDU): same as FPCDU, but for the aft sensors

3. The IR Switch Integrator Circuit (IRSIC): it manages the IR Switch power received from the F/APCDU. Though a logic OR gate, it integrates the input of the four IR Switches into one digital signal

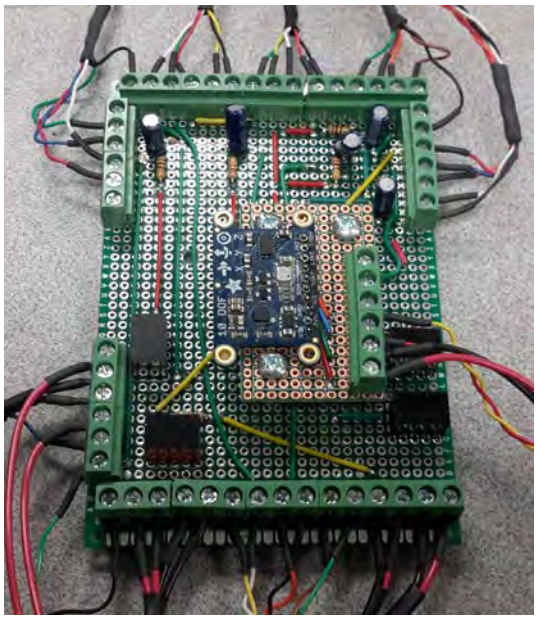


Figure 36 Electronic Board FPCDU and IMU Mezzanine.

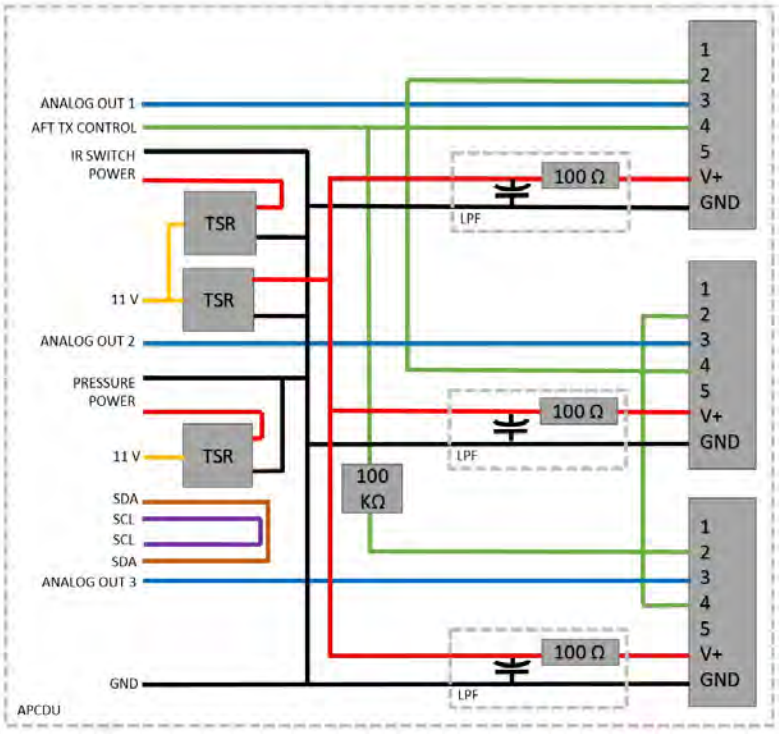


Figure 37 Electronic Board Layout (APCDU).

All the exterior electronics are installed in forward and aft compartments. Figure 38 shows the forward electronic compartment during the initial assembly, before waterproofing. Figure 39 shows the aft compartments. Both have the required space to hold junction boxes and cable runs with the required lengths to allow disassembly of the parts. As the 3D printing material absorbs water, all the electronics must be potted in a waterproof compound.

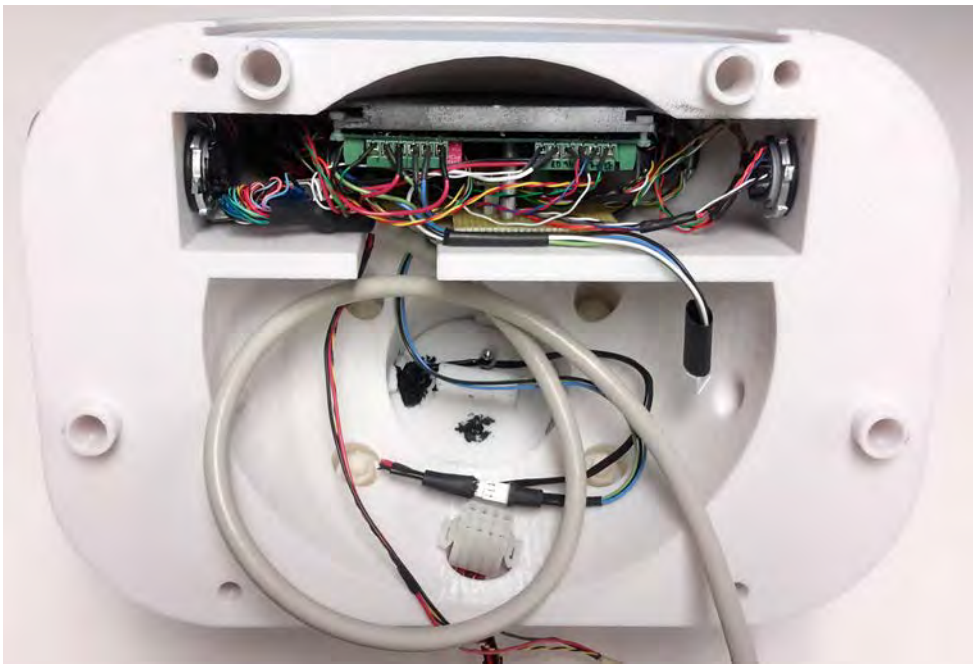


Figure 38 Forward Electronic Compartment.

Figure 39 Aft Electronic Compartment.

2. Interior Design.

The general layout is shown in Figure 30. The interior hardware design can be divided into power distribution and electronics.

a. Power Distribution.

A Polymer Li-Ion Battery Module of 11.1 [V] 5 [Ah] provides, power to all the electronics (processors and sensors) inside and outside the cylinder. This battery is installed inside the cylinder, therefore a special configuration is needed to charge the battery without opening the cylinder. This configuration is presented in Figure 40.

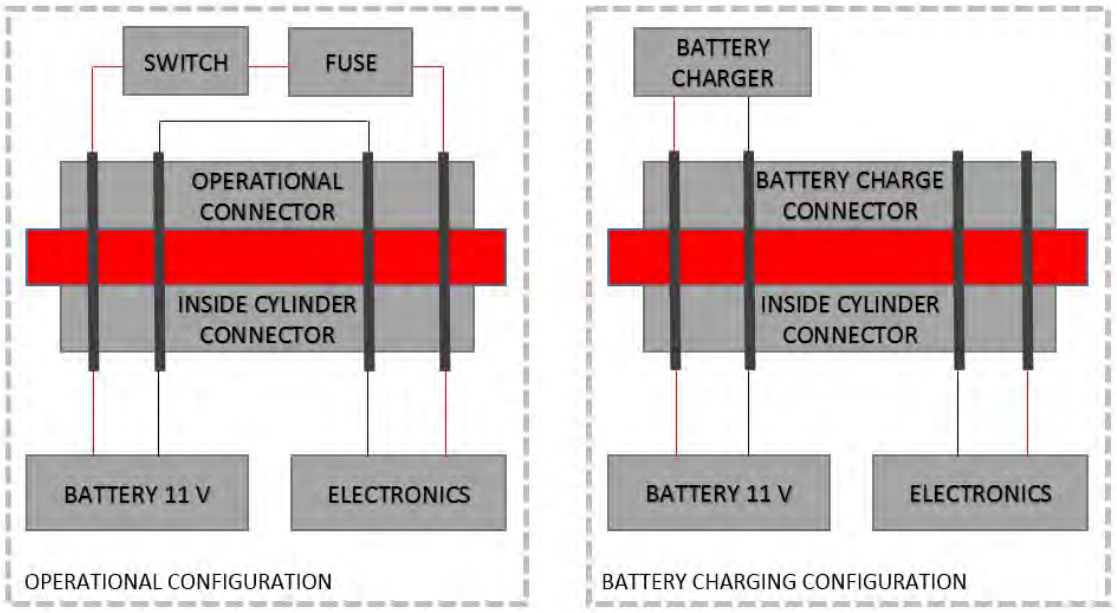


Figure 40 11 V Operational and Charging Configurations.

From the connector, the 11 V goes to a Power Distribution Board (PDB). From the PDB, the following distribution is made:

1. 11.5 V to the Raspberry Pi 2 SX300 extender board: this board has an input range from 6 to 18 V and it will supply the Raspberry Pi and its associated USB ports

2. 11.5 V for two Arduino Mega boards. The Arduino boards allows an input voltage up to 12 V. As long as the input voltage remains above 7.5 V, the boards will automatically select this input as the power source [26]. In case that the voltage drops from previous value, it will automatically switch to USB power source. For this reason, both Mega boards must be plugged into the SX300 extender board, as those ports are self-powered

3. 5 V to the 8-output relay board. This voltage powers the electronics of the relay board

4. From the Raspberry Pi's SX300 USB ports, three Teensy 3.1 boards are powered

5. 11.5 V to 6 inputs of the 8-output relay board. Table 2 summaries the output destination of each relay

Table 2 Relay Board Outputs.

Output	Destination	Natural relay position	Source of control signal
1	Forward MaxSonar array, IR Switches	Open	PP Mega1
2	Aft MaxSonar array, IR Switches	Open	PP Mega1
3	DST800 Transducer	Open	PP Mega2
4	HR100 Doppler Radar	Open	PP Mega2
5	Pressure/ Temperature Sensor	Closed	Always On
6	IMU, IRSIC	Closed	Always On
7	MC1 (closes ON circuit)	Closed	PP Teensy 3.1C
8	MC2 (closes ON circuit)	Closed	PP Teensy 3.1C

Both MC's require 24 V for operation of the land motors and tail servos. This power has to be supplied to the inside of the cylinder and distributed directly to each MC. The ON/ OFF action to the MCs are controlled by positions 7 and 8 of the relay board. Since heat extraction is accomplished by thermal conduction from the MOSFETs to an aluminum conduction plate, ventilators are installed to assist cooling.

b. Electronics

The distribution of the sensors and actuators to each preprocessor is done in the following way:

1. Arduino Mega1: The Mega board was selected for the number of available analog inputs and its 5 [V] compatibility. It processes 8 analog signals from the MaxSonar sensors and uses I2C communications with the Pressure and Temperature sensor. It also provides fore and aft array power commands to the relay board and triggers start transmission signals to the arrays. It has direct communication with the PP Teensy 3.1C (PID) through 2 digital lines. These perform Sense-Act (SA) command, to stop the vehicle when the sonars detect an obstacle below a minimum distance or

to turn off the center thruster when the vehicle is below a minimum pre-defined safe depth

2. Arduino Mega2: The Mega2 manages the serial RS232 NMEA 183 protocol for the GPS and the DST800 Transducer. It also provides power commands to the transducer and the Doppler radar. GPS power management is software controlled

3. Teensy 3.1C: its task is to perform PID control for the two land motors and three thrusters and position commands to two servos. For this it will output PWM commands through the PWM Shield. It will also perform power management control signals for the land MCs. It receives signals from the Arduino Mega1 (SA stop and inhibit center thruster) and from the IRSIC (SA stop). As the rest of the sensors and actuators works with 5 V (against the 3.3 V of the Teensy), a 4 port signal converter had to be installed. The shield's build-in prototype space was used for the converter installation. The Teensy 3.1 was selected due its superior speed and flexibility

4. Teensy 3.1C: The Teensy 3.1 was selected for its superior speed and flexibility. It was used to perform PID control for the two land motors, the three thrusters and the two tail position servos. It sends PWM commands to the PWM Shield. It also performs power management control for the land MCs. It receives signals from the Arduino Mega1 (SA stop and inhibit center thruster) and from the IRSIC (SA stop). To interface with 5 V sensors, a 4 port signal converter was used.

5. Teensy 3.1A: The Teensy 3.1 family was selected for its superior speed and memory capabilities. It was used for processing (magnetic compass calibration and Kalman filtering) accelerometer, gyro and magnetometer data, which improved IMU performance. It stores magnetic calibration data into its internal EEPROM.

6. Teensy 3.1B: Teensy is used to process output pulses from the HB100 Doppler radar. It receives Doppler train pulses through digital input 3. This allowed direct connection to an internal counter, needed for the frequency measurements

The MP task was handled by a Raspberry Pi 2 computer, connected to a SX300 board. This last board, managed power supplies and provided power to the USB port extender. It also provided with a built in WIFI antenna and a RTC device. The PPs were connected to the MP through USB ports.

For a prototype implementation, special shields were designed and constructed. Figure 41 shows the PPs and MP stacked together for integration trials.

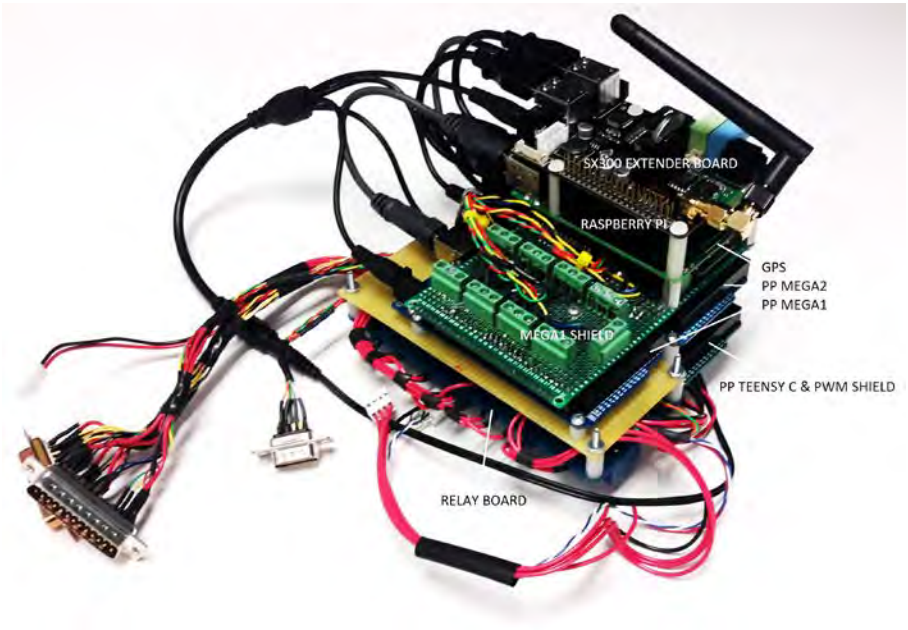


Figure 41 Interior Electronic Boards Assembly.

Figure 42 displays interior and exterior connections for preliminary integration trials. Figure 43 shows the interior electronic assembly installed inside the waterproof cylinder. Offline connectors were considered for the motor controller's setup phase. Between the cylinder's waterproof data connectors and the electronic assembly, two DB25 and one DB9 connectors are installed. This allows the user to disconnect the cylinder's covers from the electronics. Details of the wiring and interconnectors can be found on Appendix B.

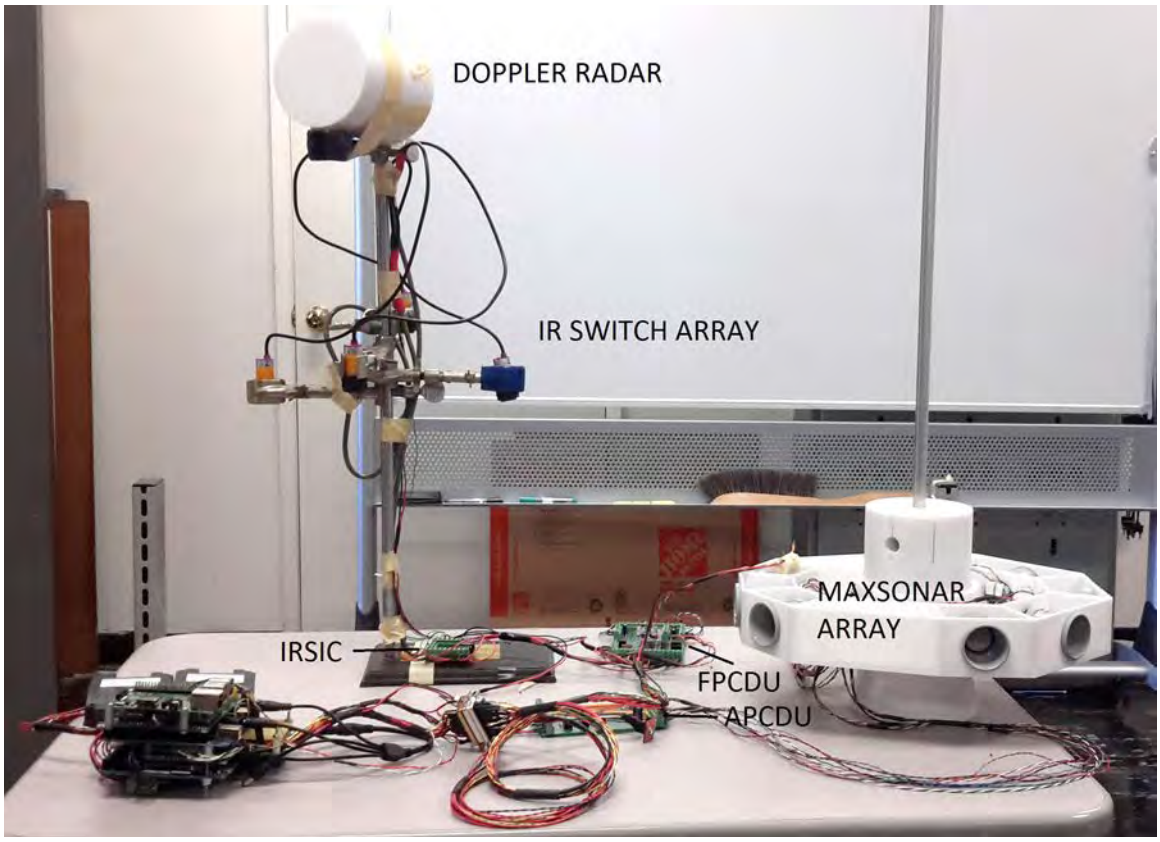


Figure 42 Interior / Exterior Electronics Connection.

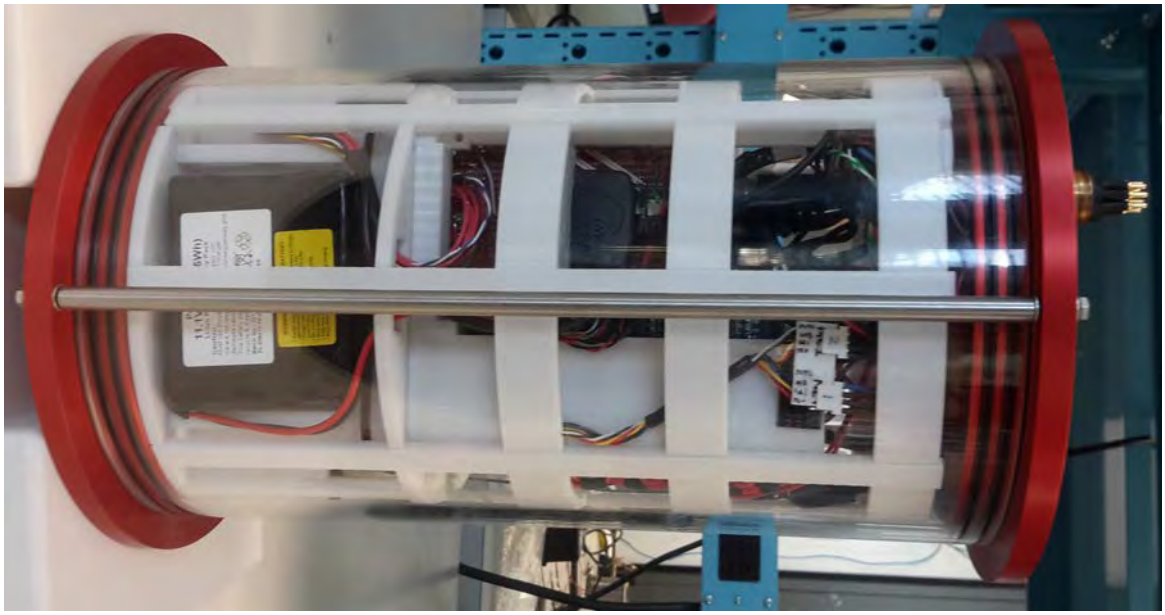


Figure 43 Interior Electronic Boards Assembly Installed Inside Cylinder.

B. SOFTWARE INTEGRATION

In this section we summarize software implementation and integration for the project. For a more technical and detailed description, please refer to Appendix C.

1. General Description

As mentioned before, the main tasks of the PP is to process the incoming data and format it in a suitable form for the use by the MP. Since the MP receives information from five different microprocessors, it is important to establish communication protocols to avoid:

1. Saturation of the MP's serial input buffer (it will slow down the program)
2. Delay in PP processing time (by transmitting data that is not going to be processed by the MP)

The above was implemented by use of a polling communication protocol between the MP and the PPs. Transmission to the MP was done by request and only the specific information requested by the MP was sent. Data transmission efficiency is maximized by avoiding the use of ASCII code. Instead, data was transmitted in binary code, with the LSB set to the required resolution. For example:

316.81° in ASCII code needs 12 bytes to be transmitted. For a binary mode transmission, the value must be divided by the required resolution (0.01°), giving a value of 31681, which can be transmitted with only 2 bytes. The selected order for the bytes is big endian (most significant byte first), hence:

$$31681_{decimal} = 7BC1_{hexadecimal} \rightarrow TX_1 = 7B_{hex} = 123_{dec}\ , TX_2 = C1_{hex} = 193_{dec} \tag{33}$$

The receiver converts this data, by summing the multiplication of each byte by the resolution value times the weight of its least significant bit:

$$\begin{aligned} Value_{Rx} &= TX2 \times \mathrm{Re}s \times LSB + TX1 \times \mathrm{Re}s \times LSB \\ &= 193 \times 0.01 \times 2^0 + 123 \times 0.01 \times 2^8 = 316.81 \end{aligned} \tag{34}$$

To manage negative values the twos-complement format uses the most significant bit to identify a value less than zero. Therefore, if a two byte word has a value more than 32768 (1000000000000000 in binary), then it corresponds to a negative number. For extracting its value, the complement number is calculated (by subtracting 65535 (1111111111111111 in binary)).

With fewer bytes to transmit, and a shorter sentence to send, less time is used. This reduces the probability of faulty message transmission. It also allows us define a fixed sentence length, and this makes it easier to parse the data.

The implementation of an efficient communication protocol allows to achieve a desired refresh rate in the main program. For land obstacle avoidance, based on experience, MaxSonar array refresh rate and an estimated maximum ground velocity of 0.5 m/s, refresh rates between 1 and 40 Hz are required.

2. Pre-Processors Software

Table 3 lists all software used with the different PPs. As stated before, all the software is written in the C language. The software is shown on Appendix C.

Table 3 Pre Processor Software.

Pre Processor	Software Name	Outputs
MEGA1	AXV_Mega1	Distance to obstacles and or depth and Temperature SA (distance and depth) signals directly to PID PP Power control signal for forward and aft sensors array (MaxSonars and IR Switches)
MEGA2	AXV_Mega2	GPS and/or Eco sounder data Prepares data in order to avoid MP waiting time Power control signals for Eco sounder and Doppler radar
TEENSY 3.1C	AXV_PID	PID for land, sea, depth, angle commands for tail deployment PWM outputs to motors, servos and thrusters Power control signals for land motor controller SA acting (stop motors)
TEENSY 3.1A	AXV_TeenA1	Kalman filtering and data fusion (Magnetometer, accelerometers and gyroscopes). Magnetometer compensation.
TEENSY 3.1B	AXV_TeenB	It will output the average and instantaneous velocity over ground of MOSARt.

Each program performs preprocessing tasks, until a command is received from the MP. For this, the program is set to a *while loop*, until the corresponding number of bytes are received by the serial port. Then the program will check for incoming bytes. Depending of the combination of bytes, the PP will perform the requested command (turn ON/ OFF sensors, transmit requested data, updates flags, etc.).

For the AXV_TeenA1, an open source program *RTIMU* by Richard Barnett has been adapted. These include new functions that address protocol data

transmissions and some minor adjustments regarding the rate of integration. The techniques implemented on the *RTIMU* are derived and explained in chapter 4.

3. Main Processor Software

Python 3 was used to program the Main Processor. The program manages generalized routines and calls sub programs for specific tasks. The description of the subroutines that handle these tasks is addressed in Appendix C. A short description is explained next:

a. AXV_sensors.py

This program polls each sensor for data. Its functions are:

1. MaxSonar: requests data from the sonar arrays and pressure sensors. It also manages power of the sensors and SA flags
2. GPS: requests GPS data updates. Manages power to echo sounder and Doppler radar
3. Sonar: requests information to the DST800 transducer
4. IMU: requests attitude information
5. Doppler: request raw velocity over ground to the Doppler radar
6. Imumaxsonar: a function that integrates MaxSonar and IMU subroutines, to decrease time delay between requests

b. AXV_actuators.py

This program moves the motors, thrusters and servos. Its functions are:

1. Motorscontrol: performs PID for the lateral thrusters, center thruster and land motors. It controls the servos and manages SA signals from the MaxSonar, Pressure sensor and IR Switch array
2. Climb: this function is not yet implemented. It sends servo commands for tail deployment

c. AXV_navigation.py

This program navigates and avoids obstacles:

1. Haversine: calculates azimuth and distance between 2 coordinates points

2. VPF: uses Virtual Potential Fields to calculate the best heading, through obstacles

3. Drspeed: calculates true bearing, the relative heading error and distance to the desired position. It uses speed and IMU information

4. Drdist: same as before, but with distance travelled as an input

5. Kalmandop: uses Kalman filter to filter the raw velocity given by the Doppler radar

6. Closesensor: actions taken to avoid obstacles at distances less than 35 cm

d. AXV_misc.py

This program performs miscellaneous tasks:

1. Initialsetup: loads the basic information of the robot, mode of operation, etc.

2. landORsea: used to distinguish between land or amphibious operations

3. Serialsetup: initializes the five serial ports used by each PP

4. Waypoint: loads waypoints from a file

5. Printout: displays function output to the terminal screen

6. firstFix: performs required tasks when obtaining an initial GPS fix

e. AXV_main.py

This program manages the main structure of the program. It calls specific functions upon request. Figure 44 shows a simplified flow diagram of the program:

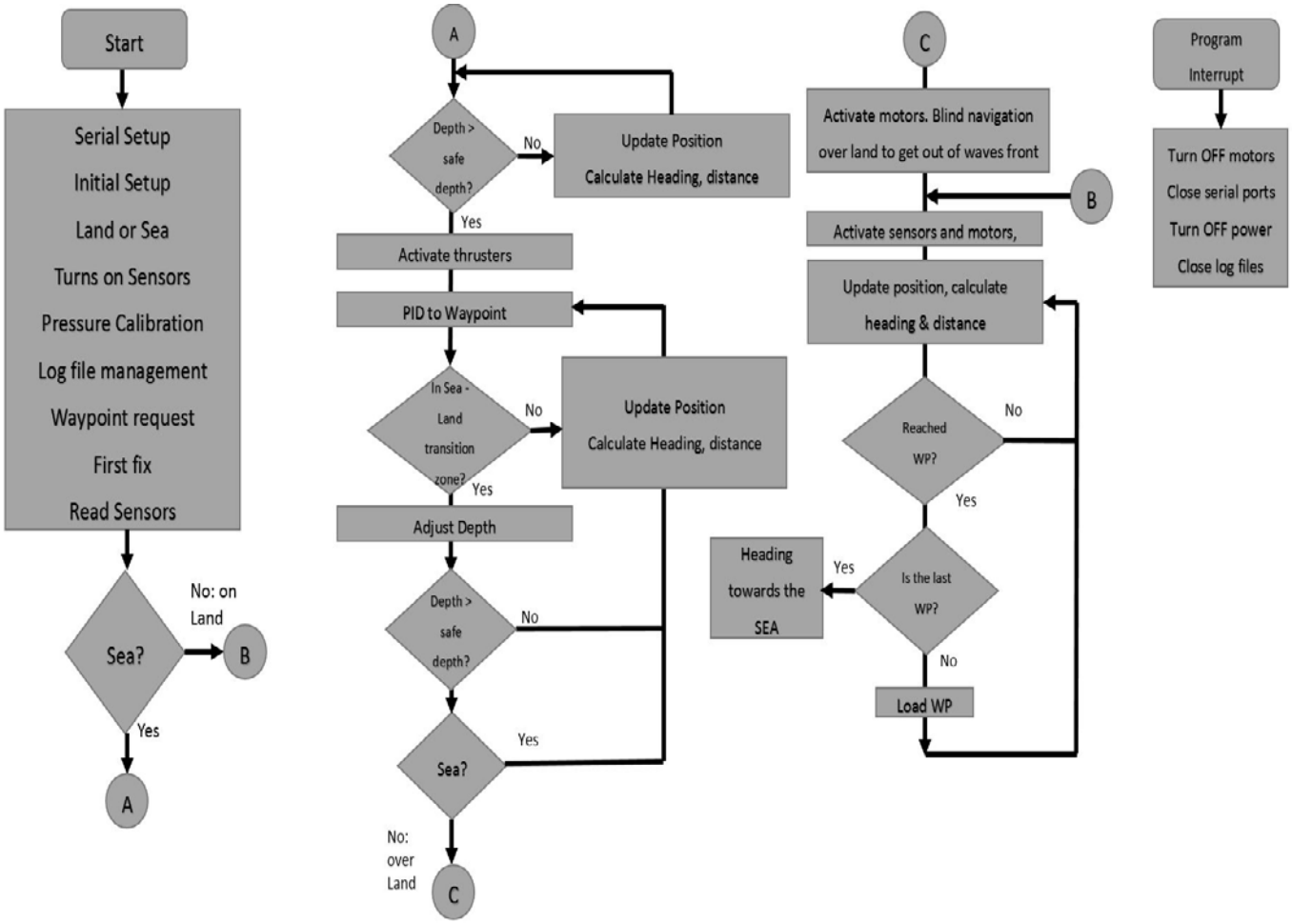


Figure 44 AXV_main.py Block Diagram.

f. AXV_functionTests.py

This is an off-line testing program. It displays menu options to perform tests on sensors, actuators and electronic functions as described below:

1. Land PID test: it tests motor PID algorithms against a pre-defined heading of 000° in 30 s runs
2. Distance DR: test the distance calculations with the Doppler radar information
3. Climbing position: tests the ability to auto position the vehicle normal to a climbing obstacle
4. Land or sea algorithm test
5. Time delay measurement of functions
6. PID and SA test: evaluates the behavior of the PID with the SA flags
7. Turn in place test
8. Obstacle climb: performed on a test platform
9. Sea PID test: checks ability to maintain a heading of 000°
10. Depth PID test: tests depth control
11. MaxSonar and IMU sensors measurements
12. Thruster's ESC calibration
13. Land Motor Controllers Setup
14. Magnetometer Calibration
15. Land Motors Controller Calibration

IV. ALGORITHMS

A. DR DISTANCE

Ground velocity measurements were conducted with the CW Doppler radar. Bench tests performed in chapter 2 give acceptable results, but the measurements were done under ideal laboratory conditions. MOSARt's own vibrations and terrain irregularities demands a more robust DR. This was accomplished with a first order Kalman filter.

1. The Kalman filter

The Kalman filter is a versatile algorithm that reduces unwanted signal noise. It improves the estimation of the state variable prior to its measurement and also has data fusion capabilities.

Figure 45 represents the computational process of the Kalman filter. The only input is the measurement, called z_k, and its final output is the estimate $\hat{x}_k$ of the state variable x_k, the physical quantity of interest.

The first step is to *predict* the estimate $\hat{x}_k$ and the error covariance (P) for the next time step (superscript "-" indicates predicted variables). The prediction is derived from the system model (related to variables A and Q, which will be explained later) and the estimated variables $\hat{x}$ and P from the previous time step [31].

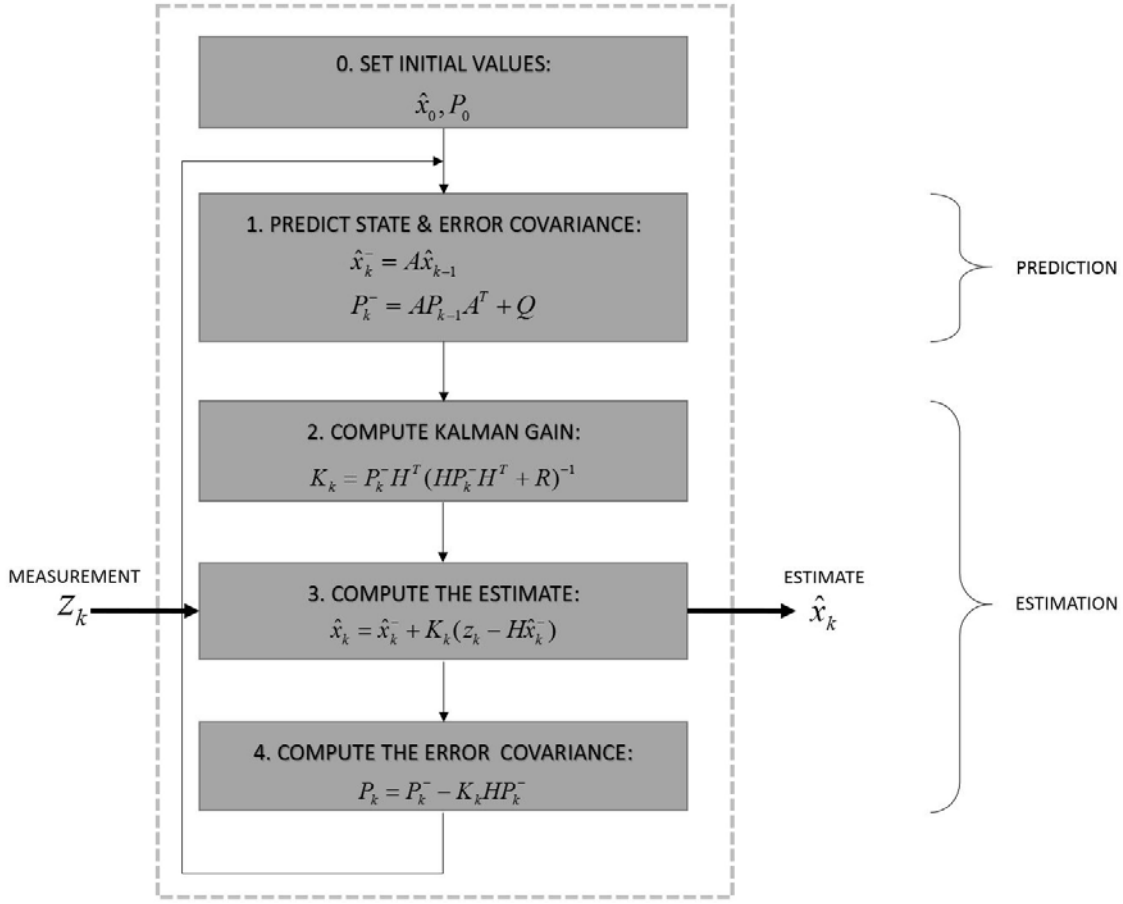


Figure 45 Kalman Filter Algorithm.

Source: [31] P. Kim, *Kalman Filter for Beginners with Matlab Examples*. Korea: A-JIN, 2010.

Steps two through four correspond to the ***estimation*** phase, and is based on:

1. The measurement (Z_k)
2. The predicted phase output
3. The system model variables *H* and *R*
4. The Kalman filter gain K_k: it relays mainly on the system model and the predicted error covariance *P*
5. The calculation of the error covariance for that loop (P_k): this represents the accuracy of the estimation process: it evaluates the difference between the estimate from the Kalman filter and the true

but unknown variable [31]. This is possible because the state variable and its estimation are related by:

$$x_k \sim N\left(\hat{x}_k, P_k\right) \tag{35}$$

Equation (35) implies that x_k follows a normal distribution with mean $\hat{x}_k$ and covariance P_k. This allows the Kalman filter to perform a probability distribution of the estimate of the variable x_k and selects the one with the highest probability as the estimate.

The *system model variables* A, H, Q and R, are the key elements for the Kalman filter performance. A and H are used in the *linear state space model*:

$$\begin{aligned} x_{k+1} &= Ax_k + w_k \\ z_k &= Hx_k + v_k \end{aligned} \tag{36}$$

The first equation of (36) represents the *state equation* and the second represents the *measurement equation* of the *system model*. As mentioned before, x_k is the state variable and z_k represents the measurement. v_k is the measurement noise and w_k the state transition noise. This is applicable as long as the noise is white noise with a normal distribution. The state transition matrix A describes how the system changes in time and H is the state-to-measurement matrix.

Q and R are diagonal matrixes that represent the covariance (variance of the variable) of the state transition and the measurement noise, respectively. These variables can be theoretically calculated or they can be tuned with experimental data. As Q is directly proportional to K (refer to equations of step 1 and 2), increasing Q will give a higher weight to the measurement z on the other hand, a lower Q will output a more stable signal, less affected by the measurement. In the case of R, it is indirectly proportional to K.

2. System Model for DR Distance Measurement

In simple terms, displacement corresponds to the velocity multiplied by the elapsed time. Nevertheless, noisy velocity measurements will cause the displacement to drift and accumulate over time. The state variables are [31]:

$$x = \begin{Bmatrix} position \\ velocity \end{Bmatrix} \tag{37}$$

To model the displacement evolution, the state transition matrix A and the state-to-measurement matrix H must have the form:

$$\begin{aligned} A &= \begin{bmatrix} 1 & \Delta t \\ 0 & 1 \end{bmatrix} \\ H &= \begin{bmatrix} 0 & 1 \end{bmatrix} \end{aligned} \tag{38}$$

Equations (37) and (38) are applied in the state space model equations (36). Now, the first equation of (36) is expanded:

$$\begin{aligned} \begin{bmatrix} position \\ velocity \end{bmatrix}_{k+1} &= \begin{bmatrix} 1 & \Delta t \\ 0 & 1 \end{bmatrix} \begin{bmatrix} position \\ velocity \end{bmatrix}_k + \begin{bmatrix} 0 \\ w_k \end{bmatrix} \\ &= \begin{bmatrix} position + velocity \cdot \Delta t \\ velocity + w \end{bmatrix}_k \end{aligned} \tag{39}$$

The first term describes the mathematical expression of the future position, hence the system noise is not included in this expression. The second term describes a constant velocity modeling for the MOSARt, even though external forces, like friction, are involved. All the velocity variations are accounted in the system noise variable. This choice of modeling was adopted as the friction and other external forces are not constants throughout the displacement.

When we expand the measurement equation, it shows that the measured state variable is the velocity that is being affected by the measurement noise:

$$z_k = \begin{bmatrix} 0 & 1 \end{bmatrix} \begin{bmatrix} position \\ velocity \end{bmatrix}_k + v_k = velocity_k + v_k \tag{40}$$

Finally, the noise was modeled and represented on the covariance matrixes Q and R. The covariance of w_k (state transition noise) and v_k (measurement noise) were calculated in Matlab and then fine-tuned with experimental data.

3. Test Trials

Figure 46 shows data from chapter 2 and the comparison between raw distance and velocity versus Kalman filter results. The Kalman filter distance is 2% less than the real displacement. As no noise was present during these trials, the results between raw and Kalman filtered data are similar.

The Kalman filter task was assigned to the Main Processor, while the Teensy reported velocity and average velocity between requests. This option was chosen as the library for matrix operation is not yet available for the Teensy microprocessor family [36].

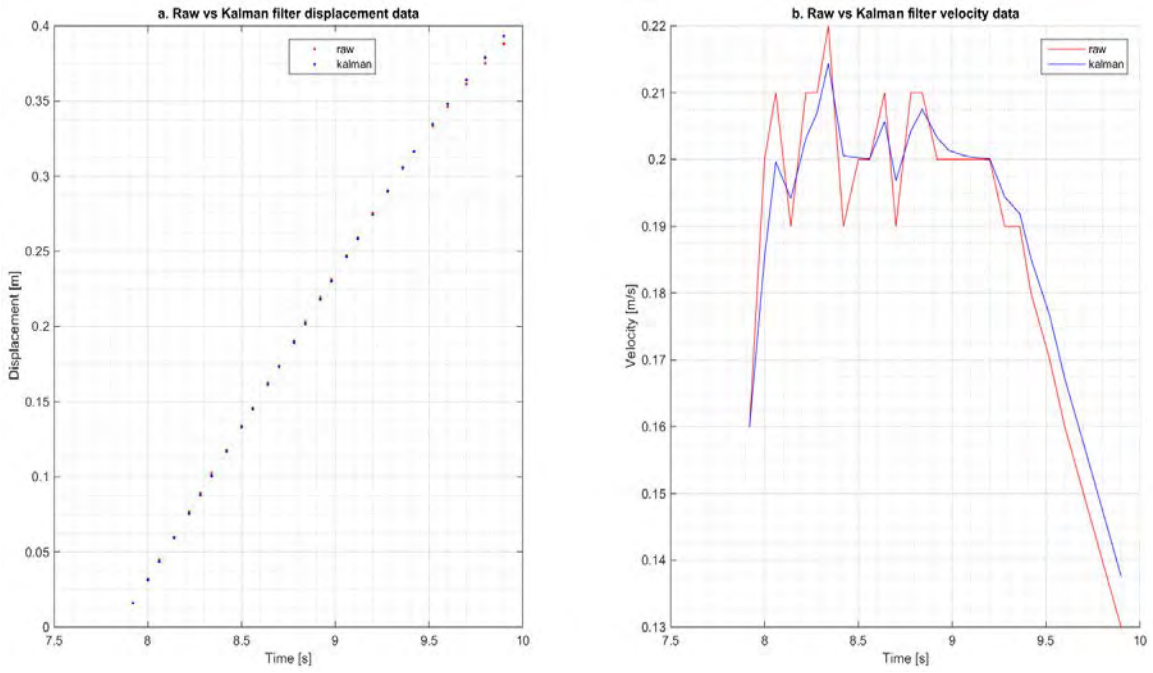


Figure 46 Raw vs. Kalman Filter Results From Laboratory Test

B. IMU FILTERING

To get reliable three-axis attitude information, data fusion was required. Angular readouts from the accelerometers, gyroscopes and magnetometers were fused and analyzed via two methods: a first order complementary filter and a Kalman filter. Prior data fusion, Gyro drift correction, Magnetometer compensation and Accelerometer noise filtering are applied. The first two actions were implemented independent of the method used to fuse data. Accelerometer noise filtering depends on data fusion. Figure 47 shows the general data fusion implementation.

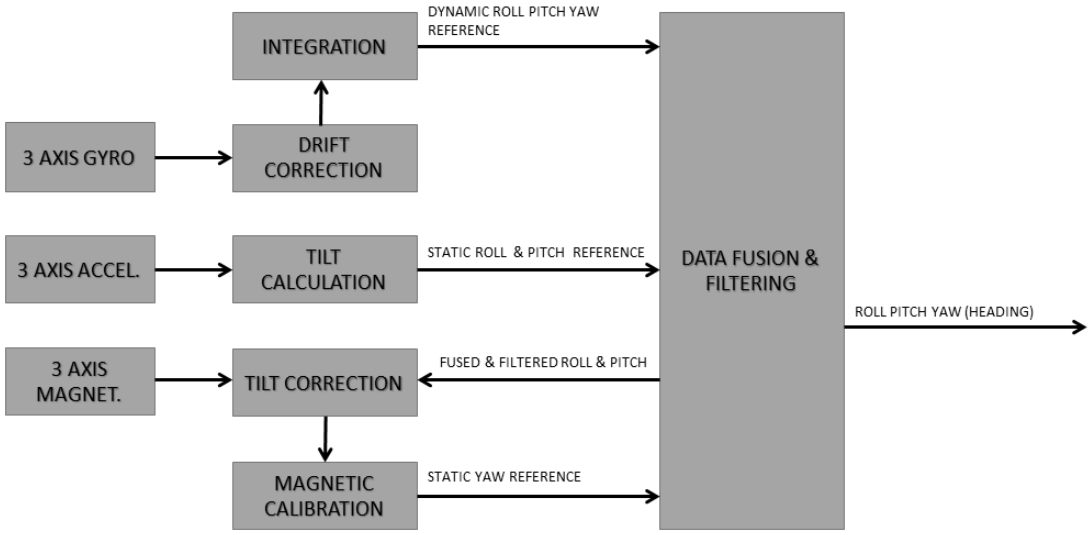


Figure 47 Data Fusion Block Diagram.

1. Gyro Drift Correction

To account for Gyro zero-bias drift, measurements were taken during preprocessor start up. This was required because temperature compensation techniques did not account for the zero-bias temperature component.

An angular rate offset was calculated for each axis, by computing a statistical mean of N samples of the Gyro rates:

$$Offset = \sum_{1}^{N} \frac{\omega_{raw}}{N} \tag{41}$$

The offset was then subtracted from the raw value. This operation was applied to all three axes.

2. Magnetometer Compensation

Before the soft and hard iron correction, the raw data must be tilt-corrected for the selected operational plane. This was necessary because the Hall sensor output is a vector in a 3 dimensional space, but the desired heading is represented as a vector in a two dimensional plane. Any component from the other orthogonal component had to disappear, and this was accomplished via a rotation transform as shown below. The rotation matrixes were employed over each Hall sensor output:

$$M_{xyz\_corrected} = R_x \cdot R_y \cdot M_{xyz\_raw}$$

$$\begin{pmatrix} M_{x\_corrected} \\ M_{y\_corrected} \\ M_{z\_corrected} \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos(roll) & -\sin(roll) \\ 0 & \sin(roll) & \cos(roll) \end{pmatrix} \begin{pmatrix} \cos(pitch) & 0 & \sin(pitch) \\ 0 & 1 & 0 \\ -\sin(pitch) & 0 & \cos(pitch) \end{pmatrix} \begin{pmatrix} M_{x\_raw} \\ M_{y\_raw} \\ M_{z\_raw} \end{pmatrix} \tag{42}$$

In (42), the reference axis was adopted according to the LSM303DLH compass reference axis. Figure 48 shows a comparison between compensated and non-compensated output heading (which was maintained fixed during roll and pitch rotations). For this results, a Matlab program *AXV_MagTilt.m* was created.

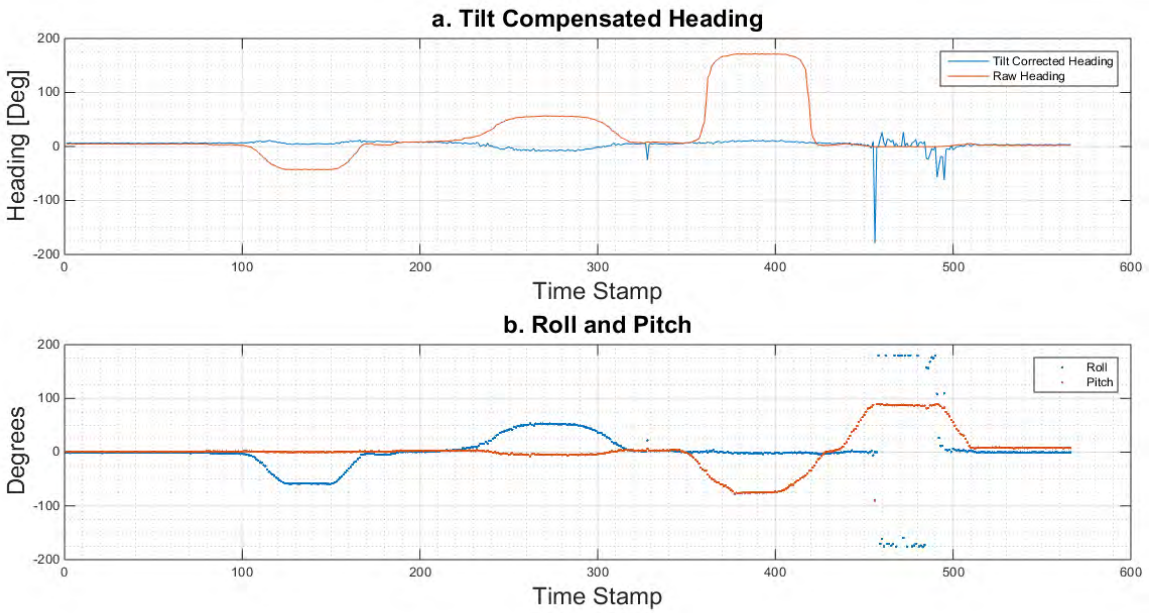


Figure 48 Tilt-Compensated and Non Tilt-Compensated Heading.

The compensated heading in Figure 48 presents a stable output during roll and pitch movements. For the roll and pitch readings, raw accelerometer outputs were used. In time-stamp 330 and 480 to 500, the heading output was unstable due to roll noise. This situation will be minimized after applying data fusion with the gyro and accelerometer data.

With the tilt-corrected Hall sensor output, calibration for hard and soft iron effects can be applied to the three planes. A Matlab program *AXV_MagCal.m* demonstrates an automated calibration process on the XY plane.

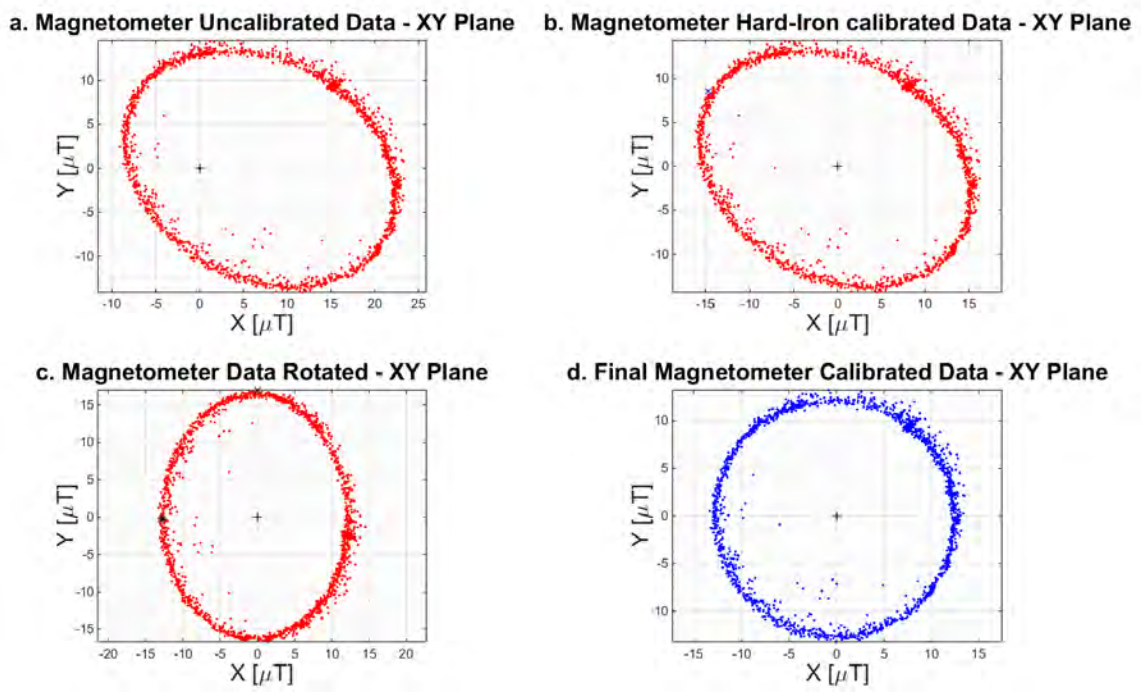


Figure 49 Magnetometer Calibration Sequence.

Figure 49 shows the procedure for calibration. First, the device under test is placed on an XY plane roundabout and raw readings are recorded during the rotation of the magnetometer. The X and Y raw Hall sensor output is seen on Figure 49.a, where hard (off center of the ellipse) and soft iron (deformation of the circle into an ellipse) effects are observed. The program detects the maximum and minimum values for X and Y outputs for the offset calculation of the data ellipse:

$$M_{x\_\text{Offset}} = \frac{M_{x\_\max} + M_{x\_\min}}{2}$$
$$M_{y\_\text{Offset}} = \frac{M_{y\_\max} + M_{y\_\min}}{2} \quad (43)$$

Figure 49.b presents the data with the offset correction. In this position, the maximum radius is calculated, along with the angle. This angle is used to rotate the ellipse (Figure 49.c), where the minimum radius is calculated (based

on a 90˚ shift from the maximum radius) and then used to calculate a scale factor:

$$S_{factor} = \frac{Semi\_Minor\_Axis}{Semi\_Major\_Axis} \tag{44}$$

The scale factor is multiplied with all the data points from the semi major axis. The result is a shifted circle that successfully factors out hard and soft iron effects as shown in Figure 49.d.

To generalize this compensation procedure the following equation can be used:

$$M_{calibrated\_xy} = \begin{pmatrix} S_{x\_factor} & 0 \\ 0 & S_{y\_factor} \end{pmatrix} \begin{pmatrix} M_{x\_raw} - M_{x\_Offset} \\ M_{y\_raw} - M_{y\_Offset} \end{pmatrix} \tag{45}$$

The first matrix represents the Soft Iron Scaling Factors. The diagonal elements correspond to the correction of an ellipse to a sphere. If no Soft Iron effects exist, we set these values to one.

After tilt-compensation and magnetic calibration, the corrected heading is calculated by:

$$Head_{corrected} = a\tan 2\left(\frac{M_x}{-M_y}\right) \tag{46}$$

The sign and the non-classical axis order accounts for the reference axis used by the magnetometer and the fact that the heading 000˚ starts at 90 ˚, with a clockwise direction.

3. Data Fusion and Filtering

As discussed in chapter 2, accelerometer data gives reliable pitch and roll output information in a near static environment (low frequency), with respect to the direction of gravity. The same result applies to the magnetometer for yaw reference. For the gyroscope, it is possible to obtain a noise free output by the

integration of the gyroscopes in a dynamic environment (high frequencies), but the outputs drift over time and lacks a reference signal.

a. The Complementary Filter

The Complementary Filter (CF) allows to merge multiple independent data sources of the same signal (in this case attitude) as long as the data's noise have complementary spectral characteristics [27]. According to [28], this fusion algorithm can be represented as in Figure 50:

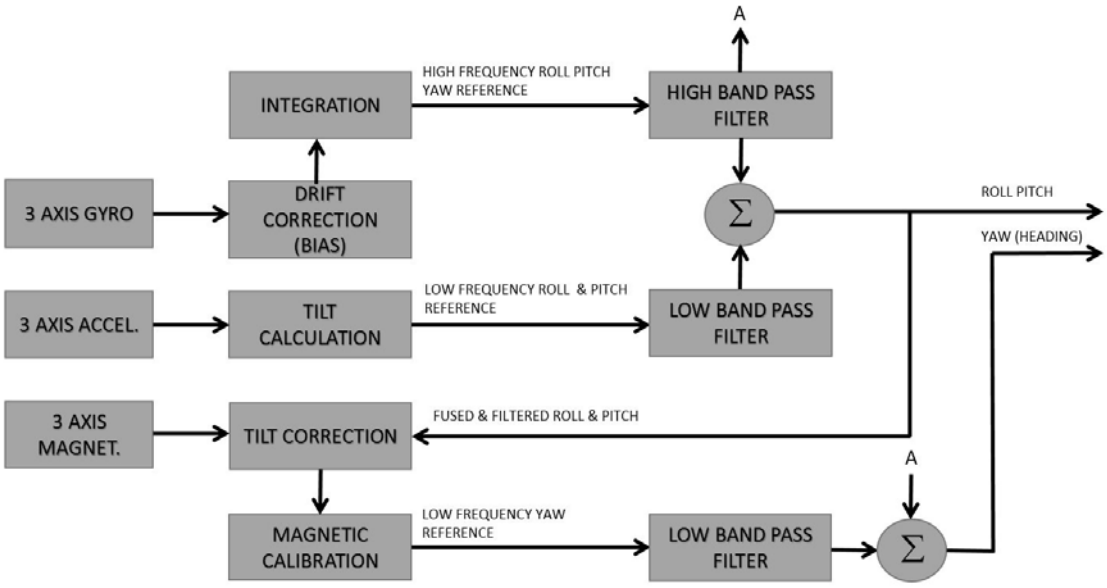


Figure 50 Representation of a Complementary Filter.

The implementation of this filter relays on the following equation for each Euler angle [29]:

$$angle_k = \alpha \cdot \left(angle_{k-1} + gyro \cdot dt\right) + \left(1 - \alpha\right) \cdot accelerometer \tag{47}$$

The alpha parameter will depend on the time constants of the system [30]:

$$\alpha = \frac{\tau}{\tau + dt} \tag{48}$$

where τ corresponds to the time constant that picks up the low frequency term and *dt* the update refresh time of the high frequency term. Usually, for a ground based IMU, an update of 100 Hz and a time constant of 0.75 s is desired This gives an approximate alpha value of 0.98. The assignments of these values are a tradeoff between noise from the accelerometer and the effects of the gyroscope's bias.

Data fusion is achieved in (47) by imposing a weighting factor for each data source: the relative low weight factor applied to the accelerometers allows that short time variations of this sensor will have a small contribution to the overall angle. On the other hand, thanks to the complemented weight factor of the gyroscope data, during these short time variations the overall angle will be mostly composed by the gyroscope, but when the angular rate is close to zero (static situation plus bias) the contribution of the gyroscope will be low. The main advantage of this filter is its simplicity and low processing demand. Thanks to the reference applied to the gyroscopes, the transformation of the body frame angular rates into the Euler angles rate of change can be avoided. Here are some of its drawbacks:

1. It does not account for accelerometer output errors due external forces (besides gravity). Even if the MOSARt application is not intended to perform aggressive maneuvering, there will be vibrations caused by irregular terrain and the Whegs wheels
2. It does not directly account for the gyroscope's drift errors

To test the algorithm, an Arduino program (*AXV_rawIMU*) was written. Gyroscope drift corrected IMU raw data is input to the Matlab program *AXV_IMUcomp.m*, where equation (47) is implemented in a simple *for* loop. Figure 51 shows the result for the roll case. The blue and magenta dots represents accelerometer and gyro-based roll angles. The noise of the accelerometer and the drift of the gyroscope can be clearly seen. The continuous red line corresponds to the fused data using the Complementary Filter algorithm, obtaining a smooth and spike-free output, but the value on steady state tends to differ from the accelerometer readings.

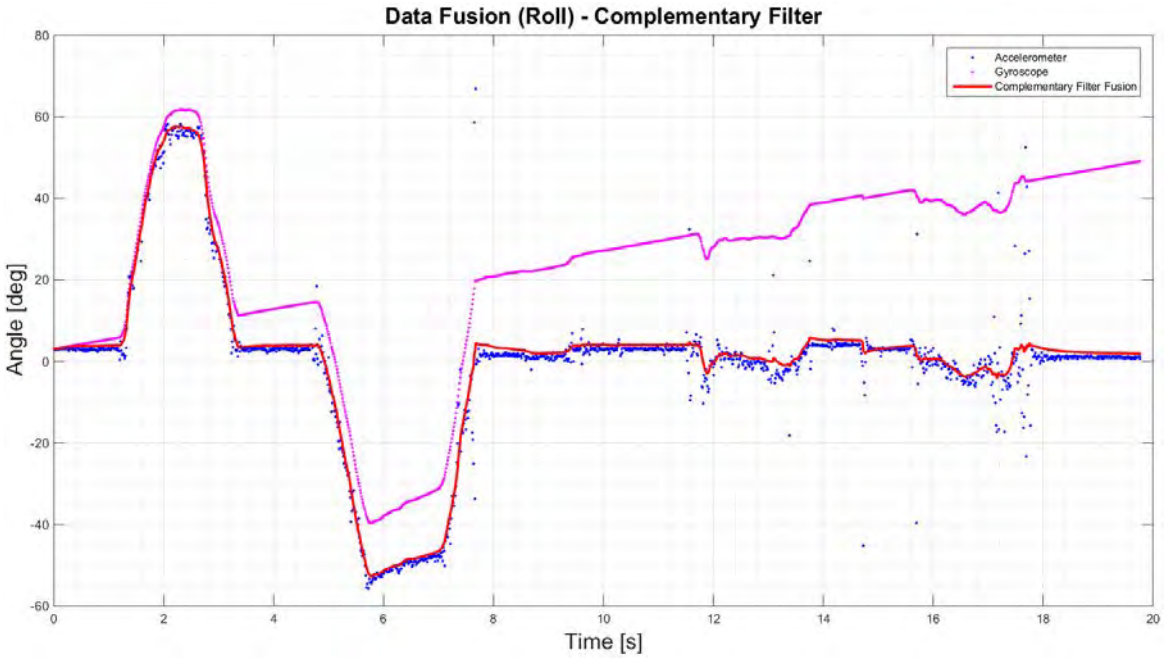


Figure 51 Complementary Filter Data Fusion.

b. Linear Kalman filtering

The theory for this filter is the same as the one explained in the previous section, although now multiple sensors will be used and data fusion will be applied.

Complementary characteristic of each device were data fused and then were operated upon by the Linear Kalman Filtering (LKF) according Figure 52.

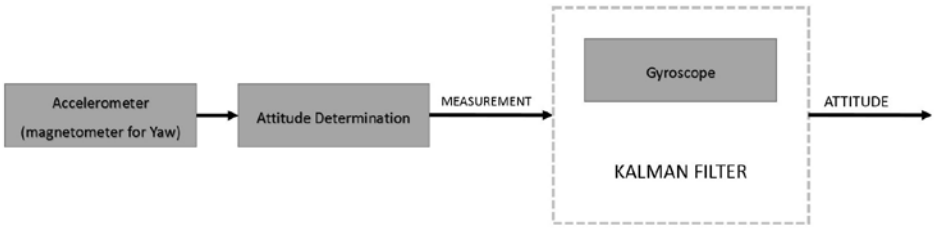


Figure 52 Kalman Data Fusion Principle.

Source: [31] P. Kim, *Kalman Filter for Beginners with Matlab Examples*. Korea: A-JIN, 2010.

Figure 52 shows that the information given by the low frequency devices are used as an input measurement. This was used to correct the gyroscope error.

The state variables for the system model are:

$$x = \begin{Bmatrix} roll \\ pitch \\ yaw \end{Bmatrix} = \begin{Bmatrix} \phi \\ \theta \\ \psi \end{Bmatrix} \tag{49}$$

The relation of the rate of change of the state variable and the gyroscope output is given by [32]:

$$\begin{Bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{Bmatrix} = \begin{bmatrix} 1 & \sin\phi\tan\theta & \cos\phi\tan\theta \\ 0 & \cos\phi & -\sin\phi \\ 0 & \sin\phi\sec\theta & \cos\phi\sec\theta \end{bmatrix} \begin{Bmatrix} p \\ q \\ r \end{Bmatrix} \tag{50}$$

In equation (50) *p*, *q* and *r* are the angular velocity measured from the gyroscope (in MOSARt's body frame). Attitude can be obtained by integration. To apply this relation to a Kalman filter state equation (related to the matrix *A*), the Euler angles must be extracted to obtain the following form [31]:

$$x_{k+1} = Ax_k + w_k \Leftrightarrow \begin{Bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{Bmatrix} = \begin{bmatrix} \ddots & & \\ & \ddots & \\ & & \ddots \end{bmatrix} \begin{Bmatrix} \phi \\ \theta \\ \psi \end{Bmatrix} + w \tag{51}$$

Unfortunately, this cannot be done. To overcome this, the state variable was changed into quaternion [33], so:

$$x = \begin{Bmatrix} q_1 \\ q_2 \\ q_3 \\ q_4 \end{Bmatrix} \tag{52}$$

A quaternion is a four-element vector (represented by q_1 to q_4 in (52)) that encodes any rotation in a 3 dimensional space. It is not as intuitive as Euler

angles, but it does not suffer from “gimbal lock” (inability to measure attitude in pitch angles near ±90°), and works well with the Kalman state equation.

To express equation (50) using the quaternion variable the following relation is used [33]:

$$\begin{Bmatrix} \dot{q}_1 \\ \dot{q}_2 \\ \dot{q}_3 \\ \dot{q}_4 \end{Bmatrix} = \frac{1}{2}\begin{bmatrix} 0 & -p & -q & -r \\ p & 0 & r & -q \\ q & -r & 0 & p \\ r & q & -p & 0 \end{bmatrix}\begin{Bmatrix} q_1 \\ q_2 \\ q_3 \\ q_4 \end{Bmatrix} \tag{53}$$

Now, a discrete integration can be applied to obtain the desired state variable. The equation is then adapted to fit the format of the state equation:

$$\begin{Bmatrix} q_1 \\ q_2 \\ q_3 \\ q_4 \end{Bmatrix}_{k+1} = \left(I + \Delta t \cdot \frac{1}{2}\begin{bmatrix} 0 & -p & -q & -r \\ p & 0 & r & -q \\ q & -r & 0 & p \\ r & q & -p & 0 \end{bmatrix}\right)\begin{Bmatrix} q_1 \\ q_2 \\ q_3 \\ q_4 \end{Bmatrix}_k \tag{54}$$

In equation (54) *I* is the identity matrix and *A* is identified as:

$$A = I + \Delta t \cdot \frac{1}{2}\begin{bmatrix} 0 & -p & -q & -r \\ p & 0 & r & -q \\ q & -r & 0 & p \\ r & q & -p & 0 \end{bmatrix} \tag{55}$$

Regarding the *measurement*, as the accelerometer (and magnetometer) attitude is calculated in Euler’s angles, it must be transformed to quaternion form:

$$z_k = \begin{Bmatrix} \cos\frac{\phi}{2}\cos\frac{\theta}{2}\cos\frac{\psi}{2} + \sin\frac{\phi}{2}\sin\frac{\theta}{2}\sin\frac{\psi}{2} \\ \sin\frac{\phi}{2}\cos\frac{\theta}{2}\cos\frac{\psi}{2} - \cos\frac{\phi}{2}\sin\frac{\theta}{2}\sin\frac{\psi}{2} \\ \cos\frac{\phi}{2}\sin\frac{\theta}{2}\cos\frac{\psi}{2} + \sin\frac{\phi}{2}\cos\frac{\theta}{2}\sin\frac{\psi}{2} \\ \cos\frac{\phi}{2}\cos\frac{\theta}{2}\sin\frac{\psi}{2} - \sin\frac{\phi}{2}\sin\frac{\theta}{2}\cos\frac{\psi}{2} \end{Bmatrix} = \begin{Bmatrix} q_1 \\ q_2 \\ q_3 \\ q_4 \end{Bmatrix} \tag{56}$$

With all the measurement variables expressed as quaternion, the measurement matrix *H* can be expressed as an identity matrix:

$$H = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{57}$$

The noise covariance matrices *Q* and *R* are set as a 4x4 diagonal matrix with their correspondence values tuned with data trials. Finally, the initial values for the state variable and the error covariance matrix are set. For this experiment, the initial position is set to the first set of Euler angles given by the accelerometer:

$$\widehat{x}_0^- = \begin{Bmatrix} q_{1\_0} \\ q_{2\_0} \\ q_{3\_0} \\ q_{4\_0} \end{Bmatrix}, P_0^- = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{58}$$

Finally, the state variables are transformed back into Euler angles [33]:

$$\begin{Bmatrix} \phi \\ \theta \\ \psi \end{Bmatrix} = \begin{bmatrix} \arctan 2\left(2(q_1 q_2 + q_3 q_4), 1 - 2\left(q_2^2 + q_3^2\right)\right) \\ \arcsin\left(2\left(q_1 q_3 - q_4 q_2\right)\right) \\ \arctan 2\left(2(q_1 q_4 + q_2 q_3), 1 - 2\left(q_3^2 + q_4^2\right)\right) \end{bmatrix} \tag{59}$$

For testing, the same raw IMU output of Figure 51 was applied to the Kalman filter algorithm and tested with Matlab. Figure 53 shows that the filter does not present drift during the steady state condition, but still produces spikes and discontinuities. These errors were attributed to the application of a linear system model to a nonlinear system: equation (50) presents the characteristic of a nonlinear system. A more adequate solution was to apply an Extended Kalman Filter.

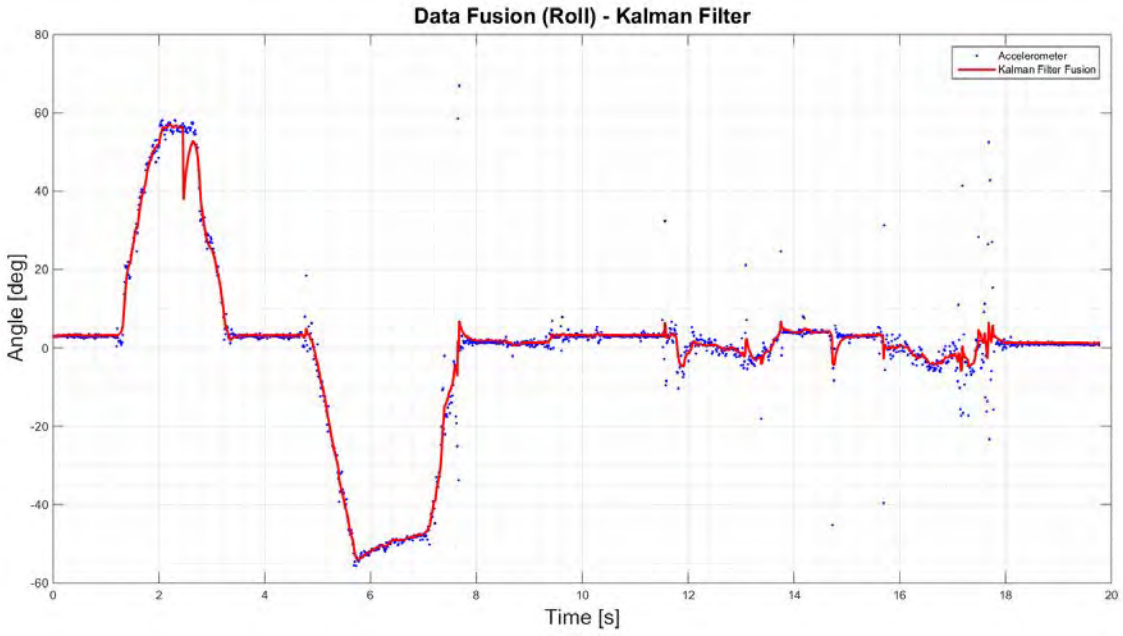


Figure 53 Linear Kalman Filter Data Fusion.

c. Extended Kalman Filter

The Extended Kalman Filter (EKF) incorporates nonlinear systems into the model. Kalman filter computational processes are shown in Figure 54. The process is the same as the first order Kalman filter, but now nonlinear functions have replaced the state and measurement matrices *A* and *H*.

$$\begin{aligned} A\hat{x}_{k-1} &\Leftrightarrow f(\hat{x}_{k-1}) \\ H\hat{x}_k^- &\Leftrightarrow h(\hat{x}_k^-) \end{aligned} \tag{60}$$

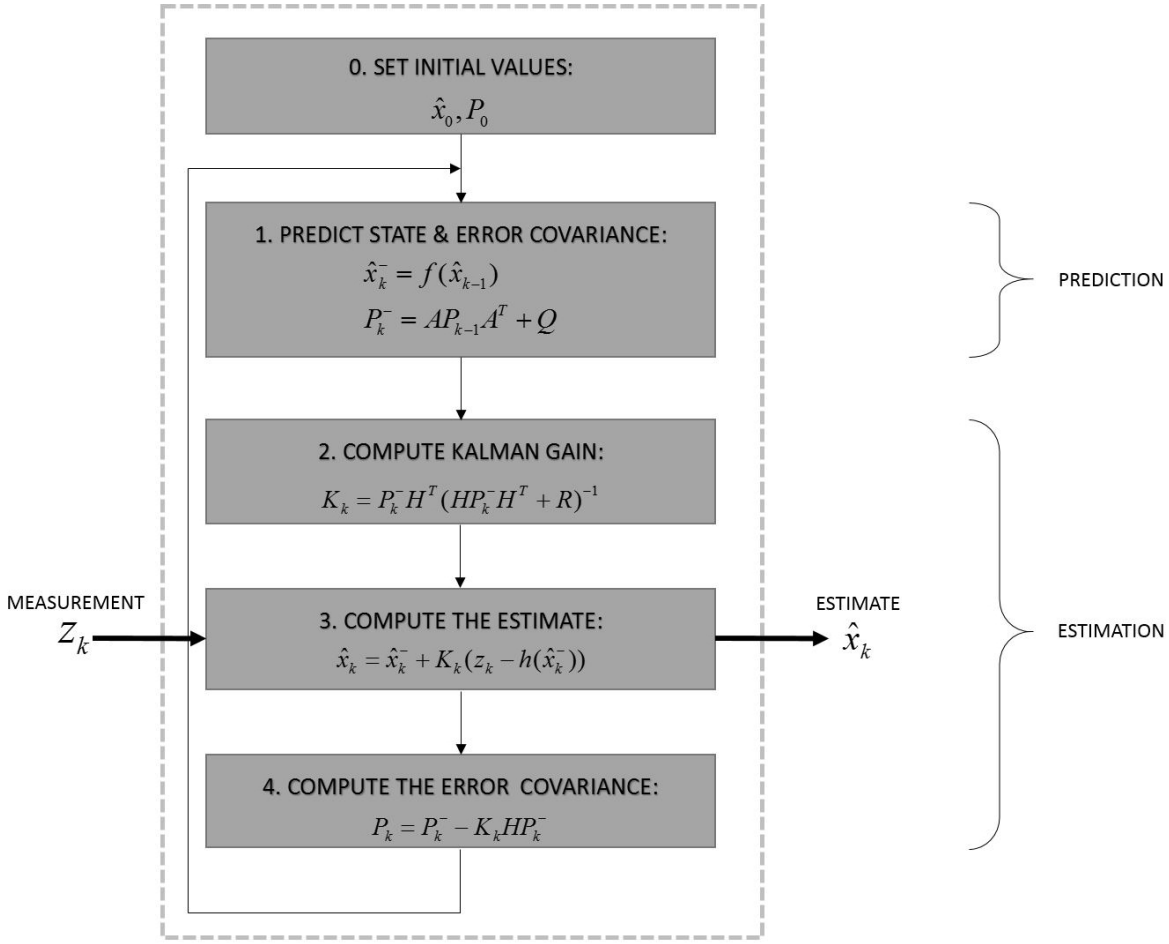


Figure 54 EKF Algorithm.

Source: [31] P. Kim, *Kalman Filter for Beginners with Matlab Examples*. Korea: A-JIN, 2010.

The functions $f(\hat{x}_{k-1})$ and $h(\hat{x}_k^-)$ shown in (60) describes the required nonlinear model. The matrices *A* and *H* are now the Jacobian of the nonlinear model (equation (61)), which allows us to linearize the system [31].

$$A \equiv \frac{\delta f}{\delta x}\Big|_{\hat{x}_k}$$
$$H \equiv \frac{\delta h}{\delta x}\Big|_{\hat{x}_k^-} \tag{61}$$

For data fusion the accelerometer data was used to calibrate the gyroscope output.

Now, equation (50) is applied to obtain the system model:

$$\begin{aligned}\begin{Bmatrix}\dot{\phi}\\ \dot{\theta}\\ \dot{\psi}\end{Bmatrix} &= \begin{bmatrix}1 & \sin\phi\tan\theta & \cos\phi\tan\theta\\ 0 & \cos\phi & -\sin\phi\\ 0 & \sin\phi\sec\theta & \cos\phi\sec\theta\end{bmatrix}\begin{Bmatrix}p\\ q\\ r\end{Bmatrix} + w\\ &= \begin{bmatrix}p + q\sin\phi\tan\theta + r\cos\phi\tan\theta\\ q\cos\phi - r\sin\phi\\ q\sin\phi\sec\theta + r\cos\phi\sec\theta\end{bmatrix} + w\\ &= f(x) + w\end{aligned} \tag{62}$$

Therefore, it is possible to use the Euler angles directly as the state variables. The measurement model turns out to be:

$$\begin{aligned}z &= \begin{bmatrix}1 & 0 & 0\\ 0 & 1 & 0\\ 0 & 0 & 1\end{bmatrix}\begin{Bmatrix}\phi\\ \theta\\ \psi\end{Bmatrix} + v\\ &= Hx + v\end{aligned} \tag{63}$$

Since the expression in equation (63) is linear, the matrix *H* is directly applied. The representation for *A* in discrete form is:

$$A = I + \Delta t\begin{bmatrix}\frac{df_1}{d\phi} & \frac{df_1}{d} & \frac{df_1}{d\psi}\\ \frac{df_2}{d\phi} & \frac{d\theta f_2}{d\theta} & \frac{df_2}{d\psi}\\ \frac{df_3}{d\phi} & \frac{df_3}{d\theta} & \frac{df_3}{d\psi}\end{bmatrix} \tag{64}$$

The results of the EKF is shown in Figure 55. A smooth, continuous and drift free roll angle is observed.

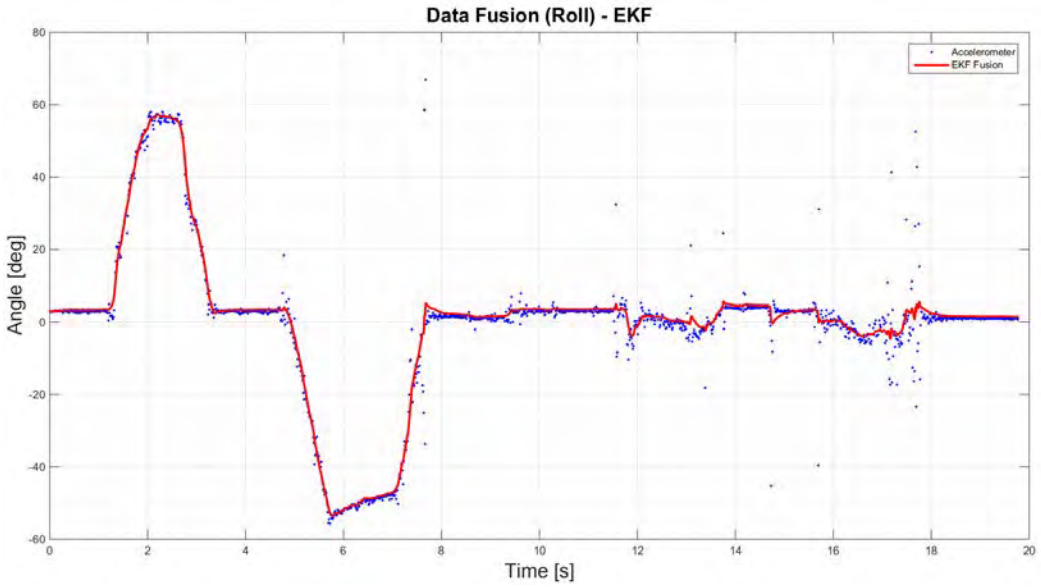


Figure 55 EKF.

Figure 56 shows a comparison between the CF and the EKF algorithms. The EKF shares the same smoothness as the CF, but it tends to follow the accelerometer readings in a better way during steady state situations.

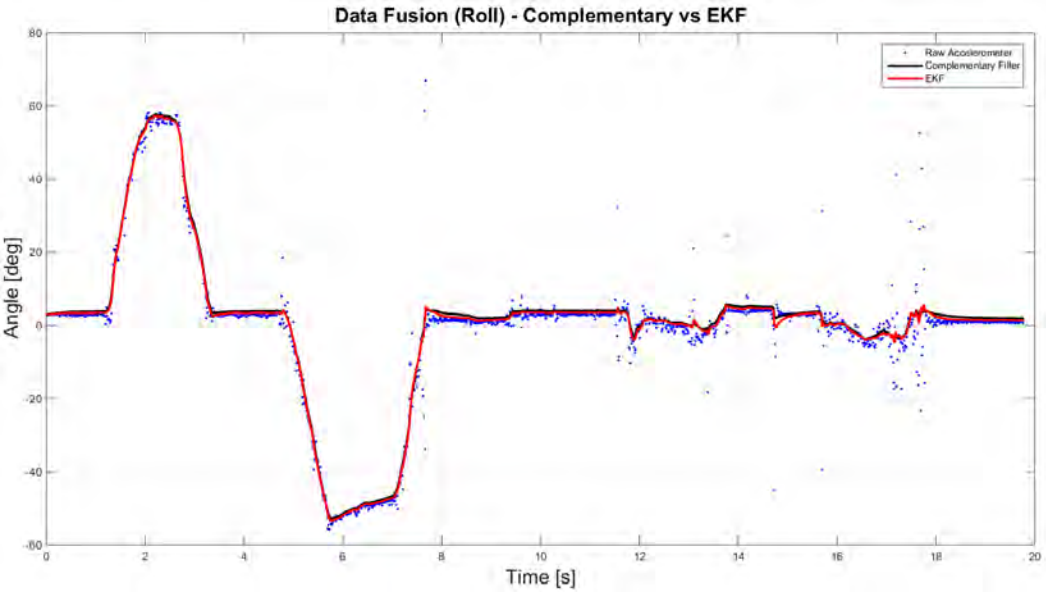


Figure 56 EKF vs. Complementary Filter.

C. VIRTUAL POTENTIAL FIELD (VPF) PATH PLANNING

VPF treats the goal position as an attractive force point source and detected obstacles as repulsive point force sources. The superposition of these forces defines the required heading.

This method is widely used in many robot applications, because it is simple [37] and produces good performance, even with inaccurate sensor data [38]. The approach taken in this thesis was to create artificial forces in a two dimensional space using input of the current waypoint (attractive) and the obstacles detected with the MaxSonar array (repulsive). From the resulting force vector, the angle is used for heading information.

1. Repulsive Point Sources

A potential that can be applied to the contacts detected with the MaxSonar sensors is given in [37]:

$$U_{rep}(\vec{q}) = \frac{1}{\| \vec{q} - \vec{q}_{obs} \|} \tag{65}$$

In the above equation, $\vec{q}$ represents MOSARt's position in the X, Y plane and $\vec{q}_{obs}$ to the detected object's position. This repulsive potential increases as the distance to the obstacle decreases. Equation (65) is suitable when each sensor detects its own contact [39], as will be the general case of MOSARt, due to the angular separation of the MaxSonars. The force vector can be calculated as follows:

$$\vec{F}_{rep}(\vec{q}) = -\nabla U_{rep}(\vec{q}) = \frac{\vec{q} - \vec{q}_{obs}}{\| \vec{q} - \vec{q}_{obs} \|^2} \tag{66}$$

The sensors are rigidly mounted on MOSARt at fixed angular positions. Consequently, equation (66) can be simplified by choosing the robot's frame as reference frame. Therefore, $\vec{q}$ can be set to zero and the decomposition of $\vec{q}_{obs}$ into Cartesian form can be obtained by using the fixed angular position of the sensors (θ_n = 0°, 45°, 90°, 135° and 180°). This allows to reduce computational

steps in the implemented algorithm. Because the sensors do not output a zero distance, there is no numerical instability in the implementation.

The total repulsion force vector is:

$$\vec{F}_{rep\_total} = \left(\vec{F}_{rep\_sensor1} + \vec{F}_{rep\_sensor2} + \vec{F}_{rep\_sensor3} + \vec{F}_{rep\_sensor4} + \vec{F}_{rep\_sensor5}\right)\alpha \qquad (67)$$

Alpha represents a repulsive gain factor, needed to "tune" the overall force vector.

2. Attractive Point Sources

To attract MOSARt to the "goal" (current waypoint to navigate), an attractive potential was implemented. In this case, the attractive potential will increase as the distance to the waypoints decreases, ensuring not to overshadow the total repulsive force. This was achieved by implementing a quadratic potential field (for short distances to the goal) and a conic potential (for distances beyond a pre-defined threshold) [39], as shown in equation (68):

$$U_{att}(\vec{q}) = \begin{cases} \frac{1}{2} \| \vec{q} - \vec{q}_{goal} \|^2 & if \ \| \vec{q} - \vec{q}_{goal} \| \le dist_{thr} \\ dist_{thr} \times \| \vec{q} - \vec{q}_{goal} \| - \frac{1}{2} dist_{thr}^{\ 2} & if \ \| \vec{q} - \vec{q}_{goal} \| > dist_{thr} \end{cases} \qquad (68)$$

In (68), $dist_{th}$ is the transition (threshold) distance from a quadratic to a conic potential. This term is also included in the conic potential to allow the gradient to be defined at the boundary between both potentials. The force is:

$$\vec{F}_{att}(\vec{q}) = -\nabla U_{att}(\vec{q}) = \begin{cases} -\beta \times (\vec{q} - \vec{q}_{goal}) & if \ \| \vec{q} - \vec{q}_{goal} \| \le dist_{thr} \\ -\frac{\beta \times dist_{thr} \times (\vec{q} - \vec{q}_{goal})}{\| \vec{q} - \vec{q}_{goal} \|} & if \ \| \vec{q} - \vec{q}_{goal} \| > dist_{thr} \end{cases} \qquad (69)$$

Beta is an attenuation factor used in the overall force calculation for MOSARt. Because the position of the goal is given in azimuth and distance, the position must be rotated 90° to calculate the X and Y components, and its direction is inverted counter clock wise. To do this we swap the sines and

cosines in the component calculations. The overall virtual force acting on MOSARt is then the sum of the repulsive and attractive forces:

$$\vec{F}_{total} = \vec{F}_{rep\_total} + \vec{F}_{att} \tag{70}$$

3. Tuning and Laboratory Tests

An *AXV_vff.m* program was created to model the calculated heading under different conditions. This allowed us to tune the gain and attenuation factor for the repulsive and attracting forces. Figure 57 shows one of the tests: The path of the robot to the desired position is saturated with contacts. Even if the position is close to MOSARt, the attractive force does not override the repulsive force of the obstacles. The course is also influenced by the attractive potential, so the robot continues to the desired position while avoiding contacts. The same algorithm has been implemented on Python.

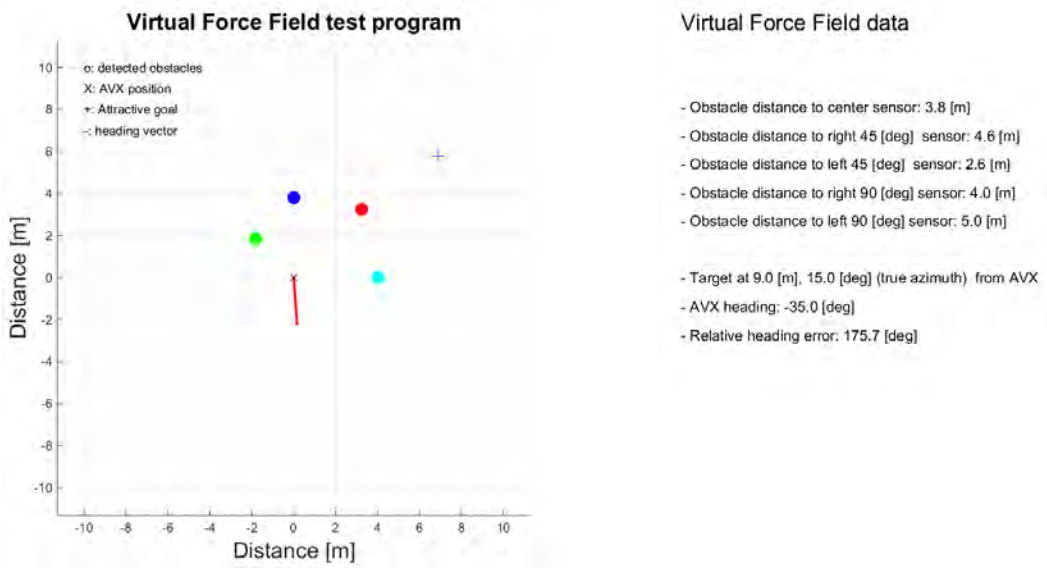


Figure 57 Virtual Force Field Test Program.

D. PID CONTROL.

PID control is a simple and flexible way to efficiently control the motion of a moving platform. It has been widely studied and implemented in previous

theses, and there is a significant amount of code available for many computer languages and platforms. For MOSARt, PID is invoked and managed via an Arduino library application [34].

1. Basic PID Theory

$$S = K_p e(t) + K_i \int e(t)dt + K_d \frac{d}{dt} e(t) \tag{71}$$

Equation (71) describes the PID control equation [35]. The signal output *S* is the control signal, which is built up from three control algorithms: proportional, integrative and derivative, each one with its corresponding tuning parameters *Kp*, *Ki* and *Kd*. The parameter ***e(t)*** constitutes the error signal: the difference between the desired goal or set point (the desired heading) and the input (actual heading). Figure 58 shows the functional control invoked for every time step.

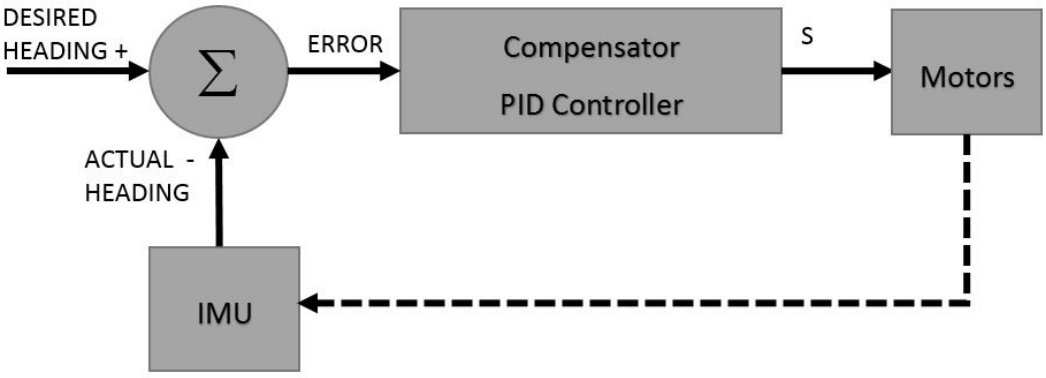


Figure 58 PID Functional Control Loop.

The selected PID library has features that allows for more flexible and reliable control, such as [34]:

a. Fixed Sample Time

User defined fixed sample times allow for more consistent PID behavior. It also simplifies the math to compute the derivative and integral portions of the controller.

b. Derivative Kick Elimination

Derivative kick is an output spike caused by changing the set point during operation. The error is the difference between the set point and the input. If the set point is changed, it will cause a step change in the error. The derivative of a step function is a delta Dirac “function” (large spike in the algorithm) and hence, a spike in the output of the PID controller. The solution depends on the following relation:

$$\frac{dError}{dt} = \frac{dSetpoint}{dt} - \frac{dInput}{dt} \tag{72}$$

Since the set point is constant (the main processor will always send the error relative to the desired heading, so the set point will always be zero), equation (72) becomes:

$$\frac{dError}{dt} = -\frac{dInput}{dt} \tag{73}$$

This also modifies the derivative term of equation (71):

$$S = K_p e(t) + K_i \int e(t)dt - K_d \frac{d}{dt} Input(t) \tag{74}$$

c. Dynamic Tuning Parameters Changes

When the tuning parameters are changed while the system is under operation, the output of the PID will suffer with an unwanted “bump”. This is generally due the classical interpretation of the integral term:

$$K_I \int edt \approx K_I \left[e_n + e_{n-1} + e_{n-2} + ... \right] \tag{75}$$

Equation (75) shows that when the K_I is changed, the new K_I will multiply the accumulated error sum. To avoid this, the integral had to be slightly modified:

$$K_I \int edt = \int K_I edt \approx K_{I_n} e_n + K_{I_{n-1}} e_{n-1} + K_{I_{n-2}} e_{n-2} + ... \tag{76}$$

The gentle modification shown in (76) ensures that adjustments done “on-the-fly” are not applied to previous error sums. This ensures a smooth transition.

d. Reset Windup Avoidance

Reset Windup is a term used to describe PID output that is outside the limits of a valid motor controller command. In this case, the integrative term continues to grow beyond the external limit. When the set point is passed back, the output of the PID has to “wind down” the violated limit. This causes delays or lags during operation.

To avoid Reset Windup, the PID library defines upper and lower limits of operation: when either limit is reached, the integration is stopped. As the proportional and derivative terms also contribute to the PID output, the overall output has to be clamped.

e. PID On/Off

If the output is simply overdriven by the operator, the PID will continue to correct, increasing its error. When the overdrive is finished and the output is switched back to PID, a coarse output is experienced. To avoid this, the library incorporates functions that allow the PID to stop computing. Additionally, to avoid bumps when turning the PID from off to on, an initialization function is also incorporated, which in general terms, updates the input to the last manual input applied (to avoid spikes due the derivative term) and the integration term is set equal to the output (as it relays in previous information).

2. Implementation Issues

Two land motors, three thrusters and two servos are used in MOSARt.

a. Land Motors

The land motors are controlled with a PID algorithm. The PID input is heading error, fed from the Main Processor. The output of the PID is tuned to the required maximum and minimum velocity command for an individual motor. The PID output is then summed to the base velocity command for each motor. On one side, this output will be added, and on the other motor, it will be subtracted,

then the required torque effect for heading steering is achieved. When MOSARt is set back to course, the PID output is zero, therefore both Whegs rotate with the base velocity.

b. Servos

The objective of the servos is to assist in climbing operations. The principle is the same as [4], but a different approach is presented in this study. The difference is that the feedback depends only on IMU pitch information. For more information refer to part E of this chapter.

c. Sea thrusters

The lateral sea thrusters (port and starboard) follow the same procedure as the land motors for the heading PID control. For the center thruster, which allows the up/ down movement underwater, closes the loop with the pressure sensor (depth of MOSARt). Additional flags are incorporated, which allows us to avoid activation of the center thruster when its intake is not under the surface of the sea.

E. CLIMBING

MOSARt is designed to climb obstacles that are 17 cm height, with the assistance of its Whegs wheels and an autonomous tail.

Climbing assumptions include:

1. A priori knowledge that it will climb an object; otherwise it will avoid it. This is accomplished during waypoint navigation setup: a flag is set for a waypoint that is required to be treated as a climbing object instead of an avoidance object
2. MOSARt must know that it is going to be able to climb the object. This will need additional sensors not included on the present design

MOSARt's tail must be pre-deployed during normal land operation. This means that as soon as it detects that is on land, it will position its tail to -180° (backwards facing).

Once the object is detected, MOSARt must position itself to the best attack heading. For this, the forward array sensors are used. In a static position, an average distance from a pre-defined number of MaxSonar readings are calculated. Then the sensor which gets the closest reading is selected, along with the following lower reading from either side of the selected sensor. From these two values, the slope is calculated, along with the relative heading error required to position itself normal to the obstacle. The heading error is corrected by turning the robot over its axis. Once the heading error is minimized, MOSARt will go forward to attack the obstacle.

The main purpose of the tails, is to raise the center of mass in a climb. The pitch of the tail is managed by use of Kalman filtered pitch information: if its value is over a threshold angle (that takes into consideration the existing pitch before climbing), a servo command angle is addressed, which will increase in each loop until the pitch value is back below the threshold. Once the obstacle is passed, as the tails are still deployed (down looking), they will cause the pitch angle of MOSARt to decrease to a negative value. After passing a pre-defined negative threshold, both tails can be commanded to be deployed to their neutral position.

F. HAVERSINE

This is an old navigation equation and is used to calculate the bearing and distance between two latitude and longitude points. Equations ((18) and (19)) where presented in Chapter 2, section II.C.2. The down side of the Haversine formula is that it considers the earth a perfect sphere.

G. LAND OR SEA DETECTION

The IMU is used to detect if the robot is over land or in the water. With motors and thruster idle, the standard derivation of the heading, roll and pitch is calculated over 4 samples, each separated 1 s. If any of the standard derivations are outside a pre-defined threshold, it is interpreted as waterborne motion.

Otherwise we assume land-based operations. The threshold must be tuned with experimentation.

H. SEA TO LAND TRANSITION

The robot is considered in the “transition zone” only when the following conditions are met:

1. Echo sounder is less or equal to 0.5 m (minimum detection depth of the DST800 echo sounder)
2. Pressure sensor depth is less than 110% of the commanded depth
3. The distance to the first land waypoint is less than 100 m

Once these conditions are met, MOSARt will decrease its navigational depth by 30%. MOSARt iterates this process until it reaches a depth barrier of 25 cm, then it will stop correcting depth height (in order to protect the center thruster). Under this condition, every 10 s the “Sea or Land” algorithm will be requested. Land detection will make MOSARt to break out the Sea loop and the land motors will be activated.

THIS PAGE INTENTIONALLY LEFT BLANK

V. TESTING AND CALIBRATION

System level tests, calibration and characterization were performed on all devices. IMU performance was evaluated in chapter 4 and its application addressed in this chapter.

A. GENERAL SYSTEM LEVEL EVALUATION

The communication system between microprocessors, the main computer, sensors and actuators were tested during the initial setup and on each individual test performed during the calibration. The main program was tested for three hours in land mode and sea mode without failure. Waypoint navigation and VPF were tested with a static scenario. Sensors and actuators information was saved to text file for logging purposes at 25 Hz. All tests were performed via a remote wireless Secure Shell (SSH) connection in the Raspberry Pi's Terminal window. Same tests, but on the Terminal window of the Virtual Private Network (VPN) connection presented a much lower refresh rate (5 Hz).

B. MAGNETOMETER SOFT AND HARD IRON CALIBRATION

The *TeensyMagCal* IMU library program written by Richard Barnett was used to calculate the magnetometer compensation factors for storage in the Teensy's EEPROM. A Python (*AXV_magcal.py*) program was written to interact with the Teensy program.

Figure 59 shows MOSARt during the procedure. All hardware was installed, except for the wheels. The position of the IMU was placed to account for the position of the batteries, motors the high voltage power supply and PWM cables.

Figure 59 MOSARt Before Magnetometer Compensation

To calibrate, MOSARt was placed in an open field and rotated all over 4π sr, until the program stopped to output updates, which indicates that the required number of samples were achieved, as shown in the screenshot of Figure 60. The python program sends the command to store the data into the Teensy's EEPROM.

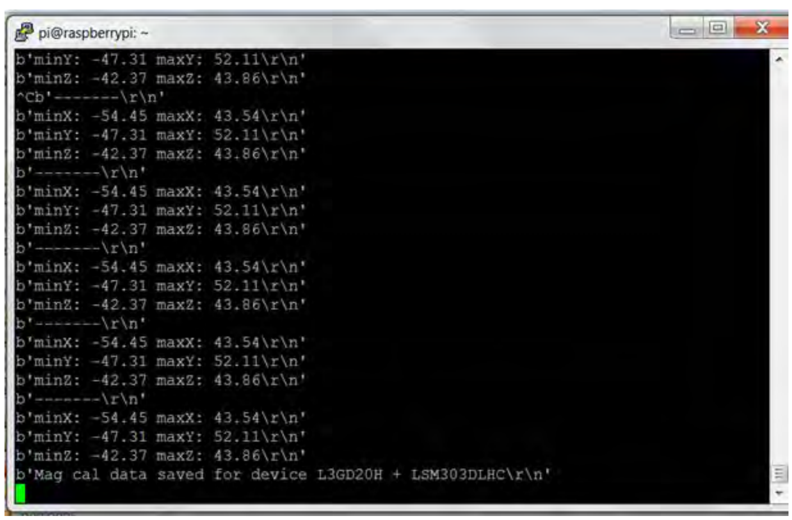


Figure 60 *AXV_magcal.py* Final Output

After the procedure, Teensy 3.1A was loaded with the AXV_IMU operational program.

C. THRUSTERS' ESC CALIBRATION

An automated ESC Thruster calibration procedure was designed with minimal user input via a *AXV_functiontests.py* – option 12 script. A series of Beep codes are transmitted by the thrusters, as detailed in [40].

Calibrations were performed with the aid of an oscilloscope (to verify proper PWM output), with successful results.

D. FUNCTIONS' TIME DELAY

The *AXV_FunctionsTests.py* – option 5 program was used to monitor time delays between functions. The results are displayed in Table 4

Table 4 Function's Delay time

Function	Remarks	Time (ms)
Forward MaxSonar Array	Turn on and report	905
Aft MaxSonar Array	Turn on and report	905
Forward MaxSonar Array	Report with SA on	8.44
Forward MaxSonar Array	Report with SA off	7.78
Aft MaxSonar Array	Report with SA off	7.73
IMU	Request attitude	14.9
GPS	Request	510
Doppler Radar	Request velocity	5.43
Kalman Filter (velocity)	First run	618
Kalman Filter (velocity)	Normal run	1.3

These time delays indicate a 20 to 25 Hz update rate for the main *AXV_main.py* program (for the land loop).

E. MAXSONAR ARRAY CHARACTERIZATION

Figure 61 shows MOSARt setup for array characterization. Two target were used: 45 cm wide (left picture) and a 9 cm diameter cylinder (right picture). While the targets were stationary, MOSARt's table was rotated. This procedure was repeated at several distances (up to 4.5 m). Data was recorded with the

AXV_functiontests.py (option 11) program. The IMU Yaw output was used for azimuth reference (in both cases, the target was around 030° azimuth).

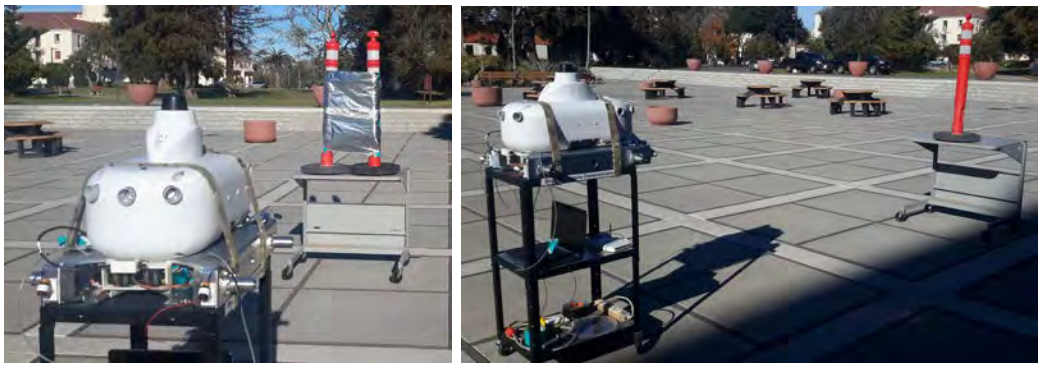


Figure 61 MaxSonar Forward Array Characterization Setup

Figure 62 shows the raw output for the small target (9 cm diameter cylinder). Each sensor lobe is represented in different colors. It displays false targets outside the cone of detection, especially for the center sensor (blue), as it is the MaxSonar that is most surrounded by other sensors. It also shows that sensors overlap in bearing for distances below 100 cm. Inspection of the data indicate that runs with distance less than 100 cm also present the majority false targets. This information was used filter the interference data, as shown in Figure 63.

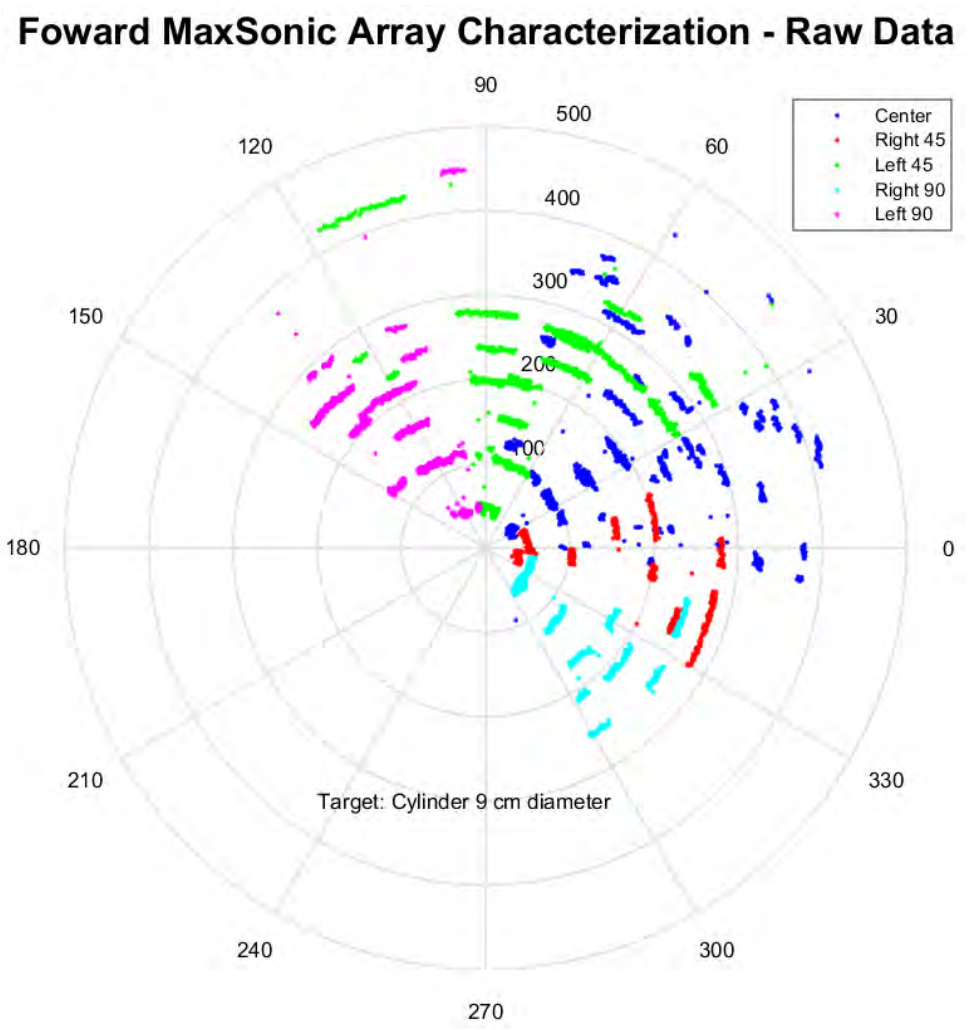


Figure 62 Forward Array Raw Output – 9 cm Diameter Cylinder

Figure 63 presents a clear view of the detection lobes of the array. As the table base used to place MOSARt could not to rotate over the center axis of the array, bearings and distance imperfections on every run must be considered. Nevertheless, for a target of this characteristic, the sensors presents the same maximum detection distance as indicated in [22], but the detection cone differs, as the one observed on the experiment shows a value of around 30°.

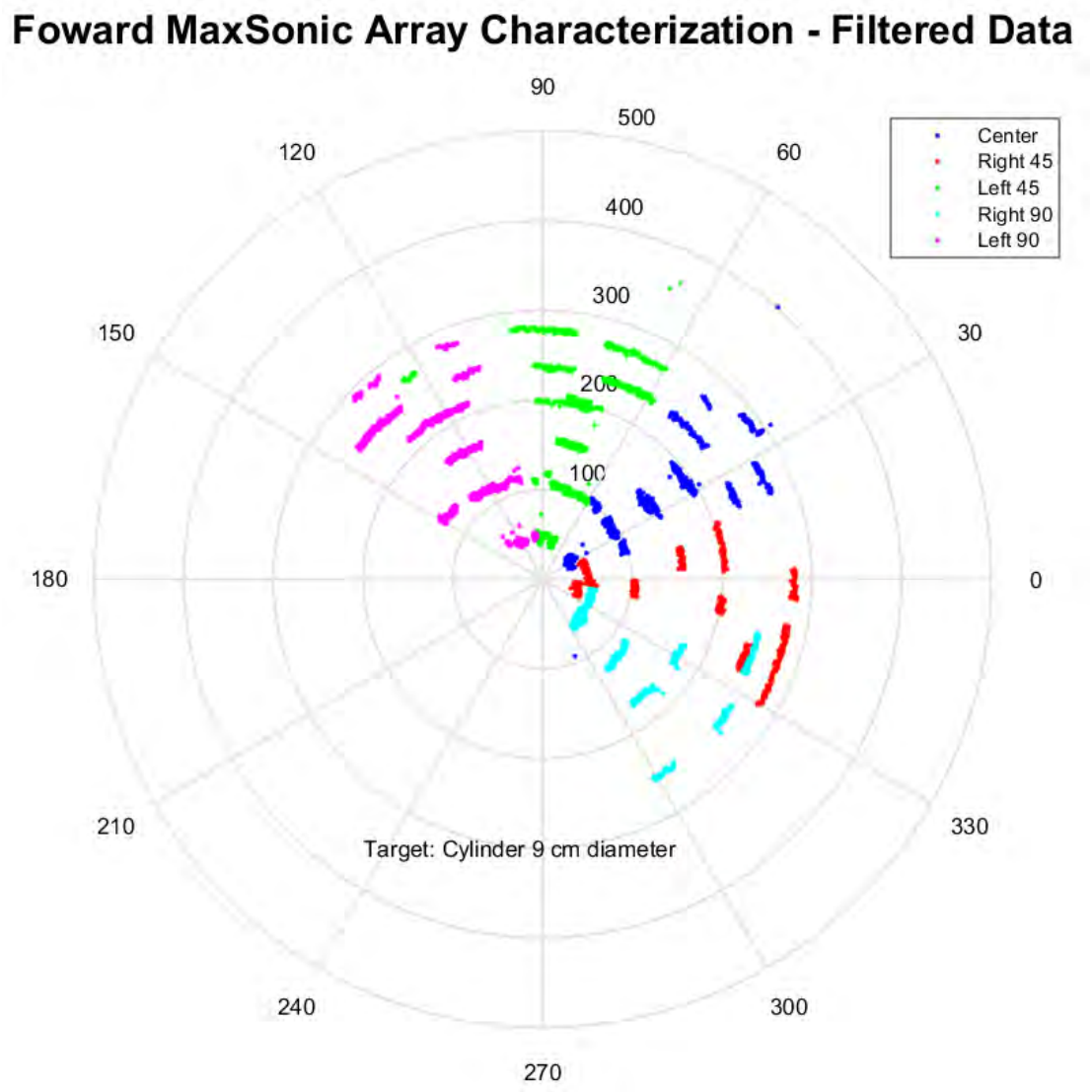


Figure 63 Forward Array Filtered Output – 9 cm Diameter Cylinder

The 45 cm raw target data is seen in Figure 64. This figure shows the raw output of the trials. For a target this size, overlap occurred at distances less than 250 cm. This effect also produced interference targets, which were filtered using the same technique as before.

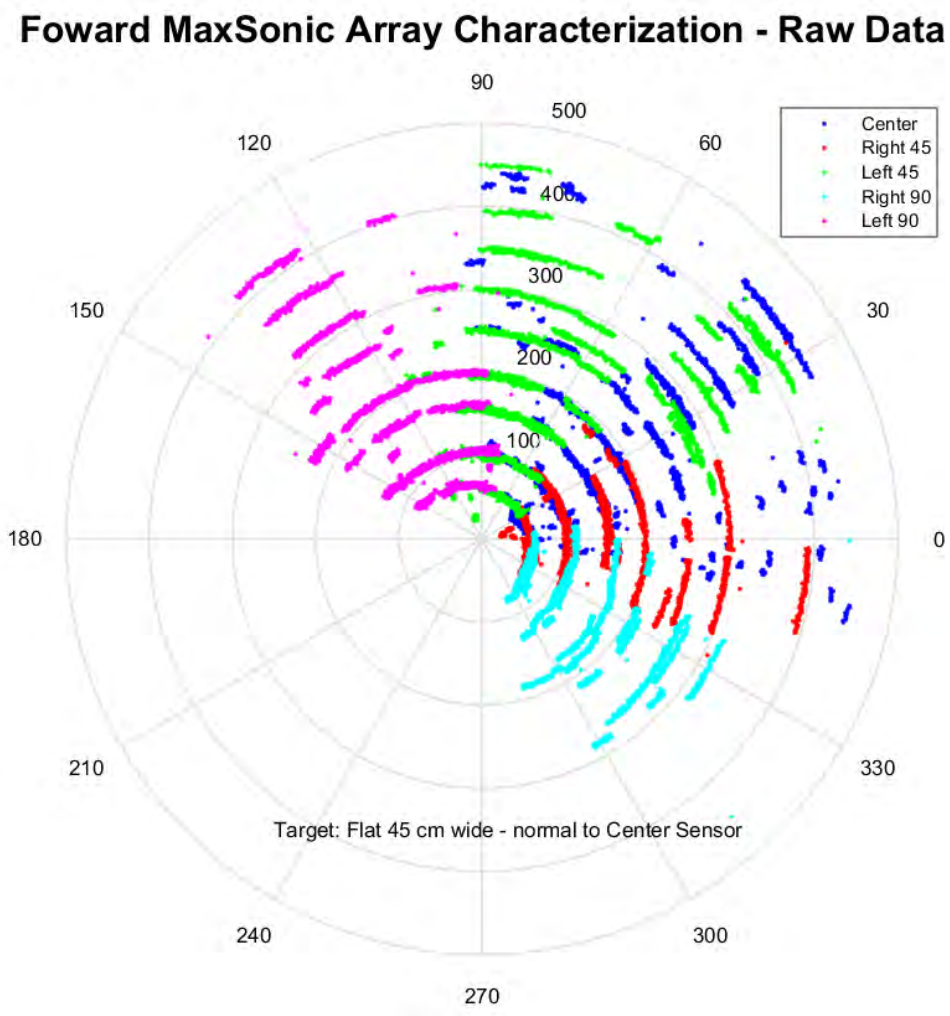


Figure 64 Forward Array Raw Output – 45 cm Wide Plate

Figure 65 displays forward array data with a distance filter applied to each runs below 2.5 m. Over this last distance. Individual lobes can be seen, with beam widths of around 15°.

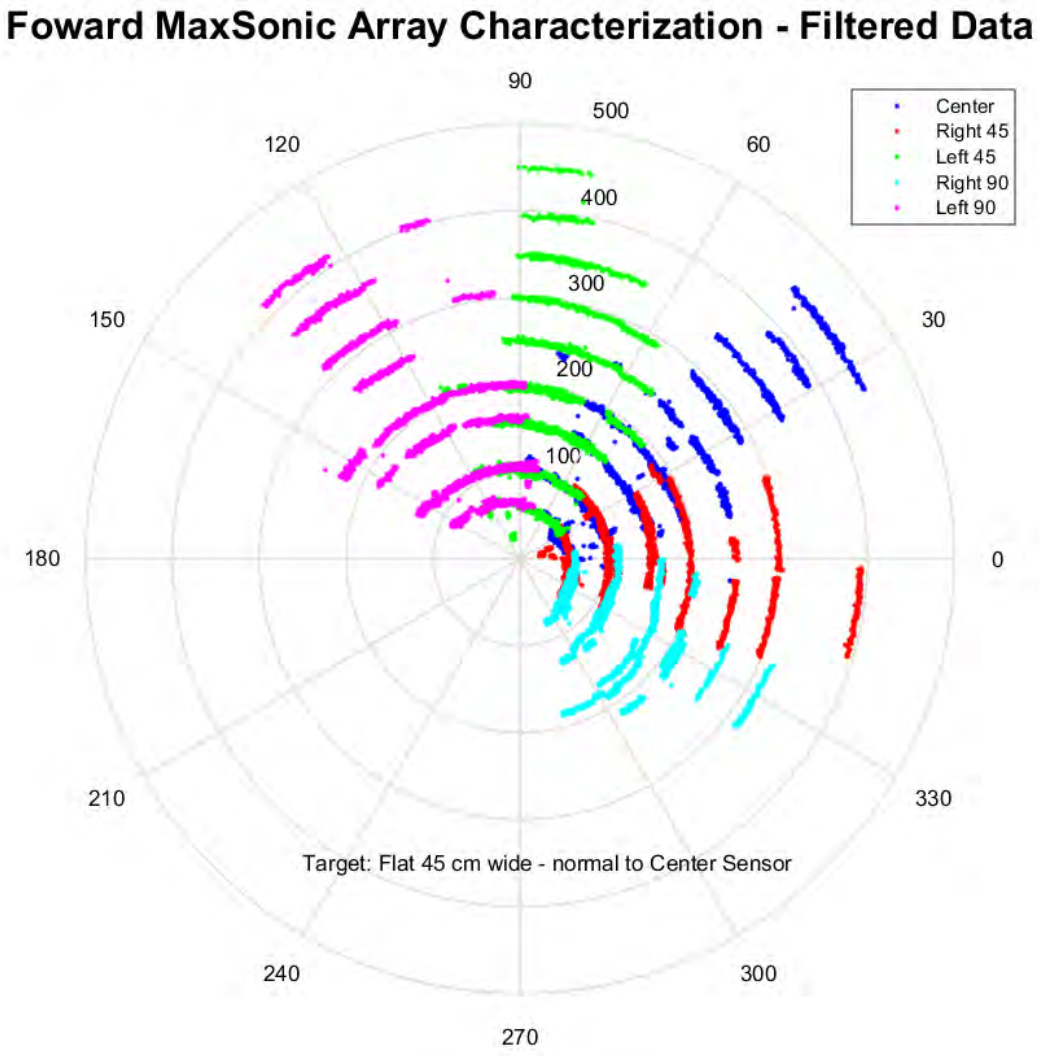


Figure 65 Forward Array Filtered Output – 45 cm Wide Plate

Conclusions:

1. The 45° array does not have blind spots

2. Overlap occurs at close distances, and is a function of target size.

3. Overlap produces interference with sensors nearby and results in sporadic false targets. False targets are detected at greater distance than the real targets

4. For a VPF application, it is best if objects are avoided at maximum detection distance, to avoid entering to the interference zone. If the target enters an interference zone, the sensor may report a false target at a greater distance. Thanks that the repulsive potential

decreases with distance, the impact on the overall heading will be less affected

F. DOPPLER RADAR AND DR

Doppler Radar was tested in an ideal (minimal vibrations) lab environment to characterize DR capabilities. MOSARt was placed on a moving table and pushed manually in a straight line for 20 m.

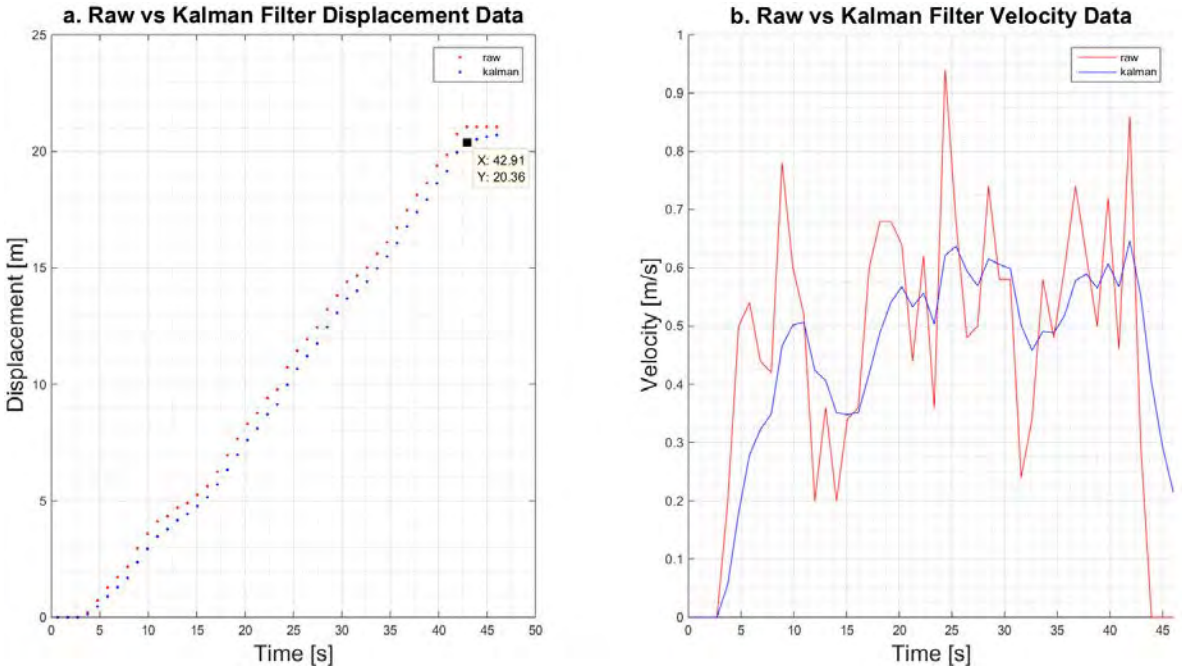


Figure 66 Doppler Radar and LKF Trial Example

Figure 66 shows one of the runs. MOSARt was stopped as soon as the 20 mark was reached, seen on the raw measurement (red) of Figure 66.a. A “hard” filter was adjusted (high value of R and low value of Q parameters). Figure 66.a shows no significant difference between the filtered and raw position data, although the LKF reached 20 m with 36 cm error against the 1.05 m of the raw data. After MOSARt was stopped, the filter continued to advance 40 cm. This is because of the hard parameters adjusted on the filter. In a real world application, the main program can correct this post-stop drift after the stop command. Figure 66.b plots the velocity of MOSARt, which shows a non-constant behavior. The

Kalman-filtered velocity (blue) shows a spike free behavior in comparison with the raw value (red).

G. MOTOR CONTROLLER SETUP AND PID INITIAL CALIBRATION

AXV_functiontests.py (option 13 and 15) was used to set up the land MCs. The Teensy control pulses were wired to the second priority pulse commands line on the MC.

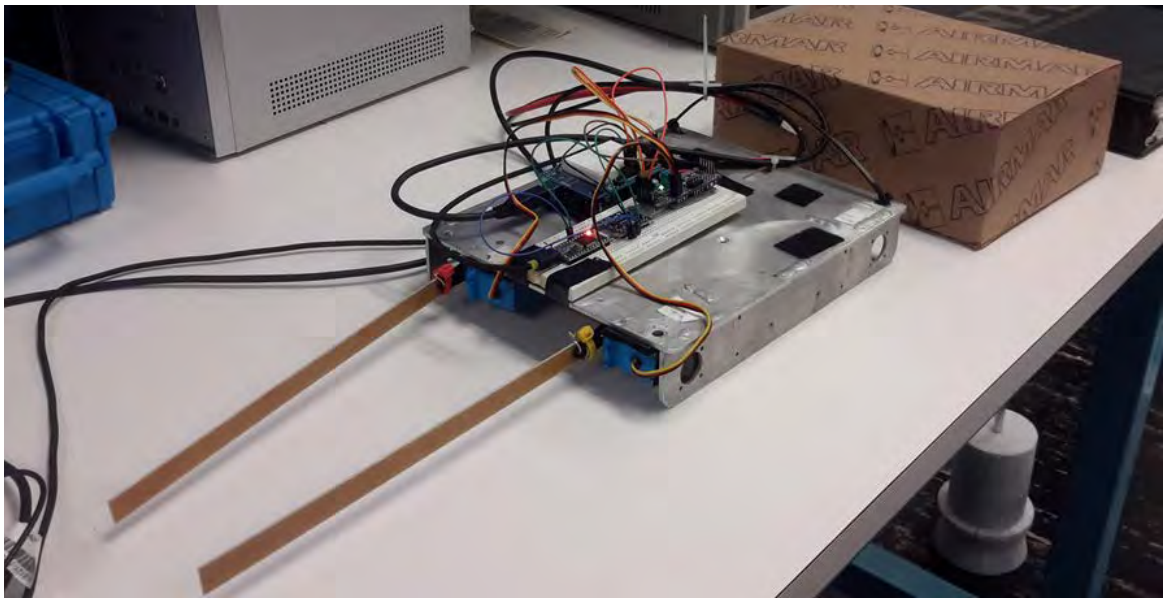
The calibration procedure is embedded in the *AXV_PID* main program. It resolves any mismatch between the microcontroller pulse commands and the MC reception. This procedure is described in the program.

Land commands were tested and base speed were adjusted. Artificial heading errors were applied to check correct Whegs rotation and velocity adjustments.

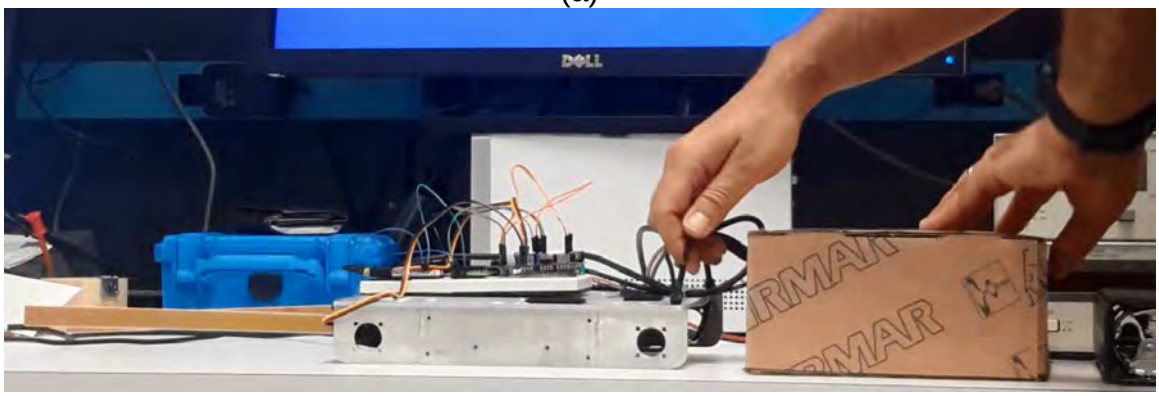
H. OBSTACLE CLIMBING

As a proof of concept, a tail and servo mechanism was implemented in the lab to test automatic reaction to pitch input for tail deployment in support land based operations. The IMU was used as the only feedback mechanism.

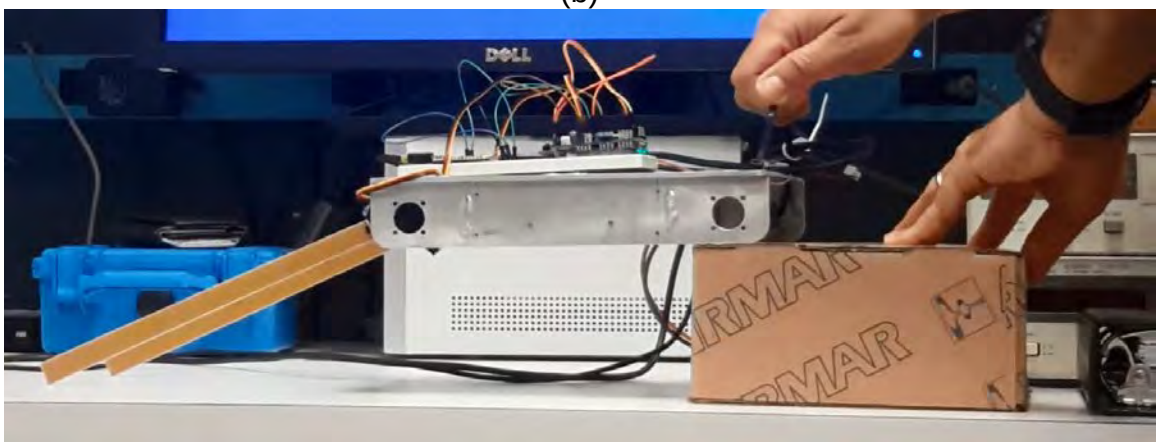
Figure 67.a shows a 25 x 22 x 4.5 cm test platform. It consist of an Arduino UNO and PWM shield. The UNO runs the *AXV_PID* software. A Teensy 3.1 is used to run *AXV_IMU* software. The Python *AXV_climb.py* program runs on an off-line laptop. The MP tasks and power supply are external to the test platform. All the components are Velcro-secured and no alignments have been made between the servos, the structure and the tail lengths.



(a)



(b)



(c)

Figure 67 Climbing Test Platform.

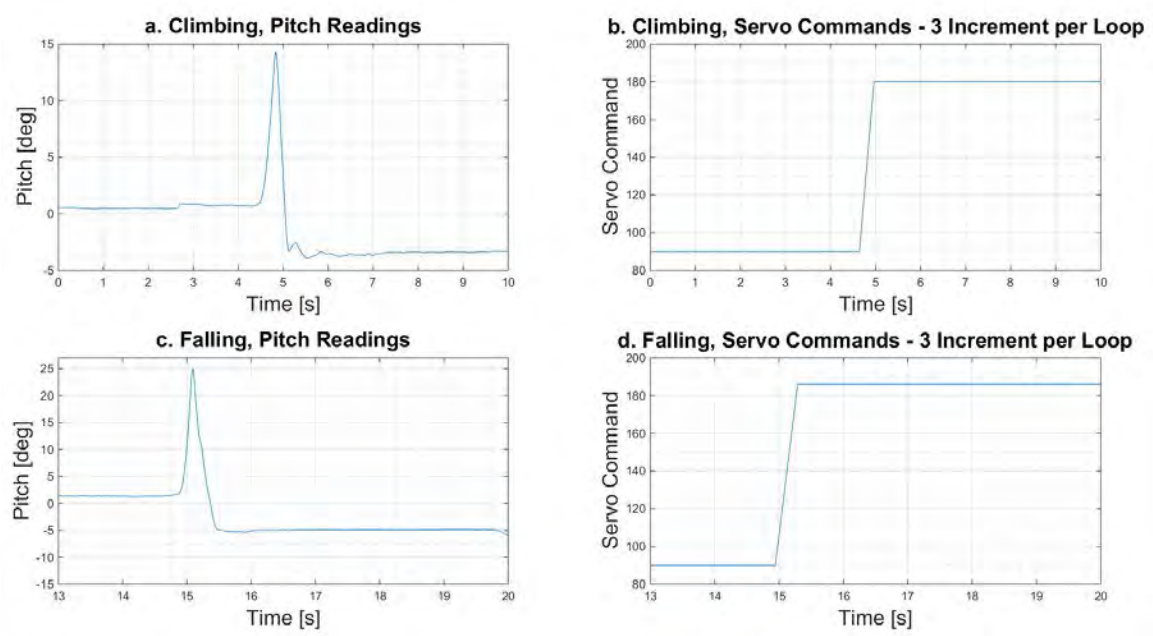


Figure 68 Pitch Correction Tests for Climbing

Figure 67.b and Figure 67.c show a sequence of tail actions during a climb. The pitch of the platform and the servo commands sent during this action were recorded every 20 ms and are displayed in Figure 68 a and b. Without the actions of the tail, the 10 cm obstacle presented a pitch of 22°. Figure 68.a demonstrates that with the action of the tail, a pitch angle no greater than 15° is reached and the time taken to correct the inclination was less than 500 ms. The test platform does not return to zero pitch after the climb because the resolution of the servo command is 3 degrees (see Figure 68.b).

Figure 68.c and d show the case when the test platform falls backward, tail first, from the obstacle. Because the tail has not reached the ground, a higher pitch angle is observed. However, once the tail reaches the ground it keeps the platform from falling or flipping over by maintaining platform horizontal pitch attitude.

I. LAND OR SEA DETERMINATION

To test in a water environment the same platform was placed on a board as shown in Figure 69. 20 trials were performed with comparative results to the land experiment.

Figure 69 Land or Sea Determination Test

THIS PAGE INTENTIONALLY LEFT BLANK

VI. CONCLUSIONS AND RECOMMENDATIONS

A. CONCLUSIONS

Surf-zone vehicle electronics, sensors and power supply were designed, implemented, constructed, integrated and partially land tested to support autonomous vehicle behavior. Terrestrial and amphibious autonomy was achieved with the integration of electronics, sensors and actuators, and managed with unique integrated algorithms.

The realization of autonomous operations included the fusion of navigational tasks, land obstacle detection, wireless communications, depth maintenance, operational environment detection and sea-to-land transition. These autonomy functions support the ISR mission and a subset of the A2/AD missions.

Physical models were used to analyze COTS components and to address limitations with signal processing. For proper Kalman filter implementation, each sub-component's physical behavior was considered and modeled. This was required to successfully improve the performance of each device.

The C programming language, along with two interpreters (Matlab and Python3), was used to program algorithms and sub programs. Tasks were divided into logical functions and these were used to simplify data fusion and to assist user understanding in support of debugging.

Integration accounted for computer and microcontroller computational limitations. The result was an efficient binary communication protocol to support our integration objectives. It is a deterministic and centralized polling communication procedure. This is the best solution for our project because the Main Processor receives information only on request. This method also frees up time for the Pre Processor to perform signal processing. The Binary data protocol is quite efficient because it is used as a technique to compress data by several orders of magnitude. The result is a 20 to 25 Hz Main Processor refresh rate.

This is more than adequate for autonomous behavior in our operational environment.

The test and characterization process for our devices, under laboratory conditions, helped us model physical behavior, understand performance and identify faults for hardware and software implementation. Dynamic field tests are still required.

B. RECOMMENDATIONS

To further the project the following are tasks are recommended.

1. Land Trials

Test the sensors and actuators in a dynamic environment. This can be conducted with the *AXV_functiontests.py code.* PID parameter calibration must be the first step in this series of tests.

2. Sea Trials

Pot the remainder of the exterior electronics in preparation for sea trials. Conduct underwater PID tuning by running the option for underwater tuning in the AXV_functiontests.py program.

3. Hardware

a. Doppler Radar

For the Kalman filter investigate the assumption that the noise is Gaussian. An EKF is a possible solution if the velocity relations are not linear. If Fourier analysis is needed, it is recommended an upgrade to a higher frequency transceiver, such as the 20 GHz K-LC1a family [41], shown in Figure 70. This device has a narrower beam width with a 50 MHz bandwidth and will support additional techniques for proper signal processing.

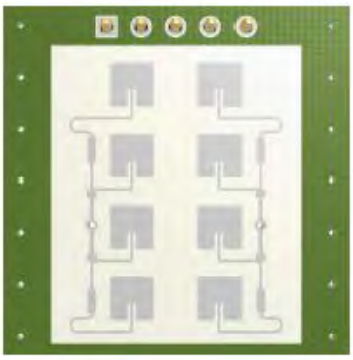


Figure 70 K-LC1a Dual 4-Patch Antenna Doppler Transceiver

Source: [41]: RFbeam Microwave GmbH, *K-LC1a_V4 Doppler Transceiver*. Available: http://www.rfbeam.ch/downloads/data-sheets/

b. IR Switch Array Performance

Laboratory tests showed that the IR Switch sensors output false “pothole” alarms. This was a function of installation height and the type of sensed terrain. A possible solution is to incorporate a counter filter (*m* out of *n* detections) in the *AXV_PID* software.

c. Outside Cylinder Electronics

Outside electronics compartments should be designed as a separate structure from the main body. This will allow easier work access to measurements, repairs and modifications.

4. Software

A simple Graphic User Interface (GUI) was implemented in this thesis. Graphics and plots were avoided to minimize computational requirements. Python has built-in web functions and can be used with an external computer to remotely view data graphs and plots “on-the-fly”.

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX A. HARDWARE SELECTION

Table 5 Requirements To Fulfill MOSARt's Objectives.

Objective	Requirement	Means to accomplish	Remarks
Maintain a desired course and depth	Depth measurement.	Pressure sensor.	Must be able to work underwater.
	Pitch, roll, heading.	IMU.	
	Proportional – integral-derivative (PID) control	Software implementation.	
Sea to land transition	Pitch, roll, heading,	IMU.	
	Sea level altitude.	Pressure sensor.	
Active positioning	Satellite navigation.	GPS.	
DR	Pitch, roll, heading.	IMU.	
	Velocity estimation.	Log.	
		Doppler radar.	
	Sea level altitude.	Pressure sensor.	
Obstacle avoidance in an efficient and simple manner.	Range estimation.	Sonic detectors.	Temperature sensor for sonic sensor compensation if required.
	Portholes detection.	IR switches.	
	Temperature	Temperature sensor	
	Virtual Potential Field.	Software (SW) implementation	
Waypoint navigation.	Satellite navigation.	GPS	
	DR	See Dead reckoning.	
	PID	SW implementation	
Tail deployment.	Pitch, roll,	IMU	
	PID control	SW implementation	
Communications	Wireless Transmit/ Receive (T/R)	WIFI	For prototyping purposes.

Objective	Requirement	Means to accomplish	Remarks
General	Computational capabilities.	Dedicated microprocessors.	For sensor's Pre-Processing.
		General purpose computing.	For MP

Now that a hardware and software techniques has been defined, hardware components have to be selected. Table 6 indicates the specific component selection.

Table 6 General Hardware selection

Task	Name of component
IMU	Adafruit 10-Degree of freedom (DOF) IMU breakout - L3GD20H + LSM303 + BMP180
GPS	Adafruit Ultimate GPS Breakout - 66 channel w/10 Hz updates - Version 3
	GlobalSat AT-65SMA GPS Antenna
Pressure sensor	SparkFun Pressure Sensor Breakout (MS5803-14BA)
Temperature	SparkFun Pressure Sensor Breakout (MS5803-14BA)
Doppler radar	HB100 Doppler Speed Sensor
Log	DST800 Transducer
Range	HRXL-MaxSonar-WR MB7360
IR switches.	Geetech Infrared proximity switch module
Motor Control	Adafruit 16-Channel 12-bit PWM/Servo Shield-I²C Interface
	SDC2160S motor controller
PP	Arduino MEGA, Teensy 3.1.
MP	Raspberry Pi model 2
WIFI	Raspberry Pi model 2 with WIFI antenna

APPENDIX B. HARDWARE

A. CABLES AND CONNECTORS

Table 7 Arduino Mega 1 Pin Out.

Mega1 Pin Out			
Pin Name	**Cable Color**	**Signal**	**To/ From**
A0	Orange	Maxsonar Forward 1	DB25F – WT26 - FPCDU
A1	Black	Maxsonar Forward 2	DB25F – WT26 - FPCDU
A2	White	Maxsonar Forward 3	DB25F – WT26 - FPCDU
A3	Brown	Maxsonar Forward 4	DB25F – WT26 - FPCDU
A4	Yellow	Maxsonar Forward 5	DB25F – WT26 - FPCDU
A5	Red	Maxsonar Aft 6	DB25M – WT8A - APCDU
A6	Brown	Maxsonar Aft 7	DB25M – WT8A - APCDU
A7	Yellow	Maxsonar Aft 8	DB25M – WT8A - APCDU
SLA	Black	Pressure sensor I2C	DB25M – WT8A - APCDU
SCL	Blue	Pressure sensor I2C	DB25M – WT8A - APCDU
D8	Green	Forward Maxsonar Array Trigger Control	DB25F – WT26 - FPCDU
D9	Green	Aft Maxsonar Array Trigger Control	DB25M – WT8A - APCDU
D10	Red	Distance SA	Teensy 3.1 C Pin 6
D11	Green	Depth SA	Teensy 3.1 C Pin 5
D12	Red	Maxsonar Forward Array Power Control	Relay Input Pin 1
D13	Black	Maxsonar Aft Array Power Control	Relay Input Pin 2
PWR Socket	Black	11 V Power	4 way splitter cable
USB TB	Gray	USB Serial Signal (Port MEGA1)	Raspberry Pi Serial Port

Table 8 Arduino Mega 2 Pin Out.

Mega2 Pin Out			
Pin Name	**Cable Color**	**Signal**	**To/ From**
TX3	Red	TX RS232 GPS	Pin RX GPS
RX3	Yellow	RX RS232 GPS	Pin TX GPS
5V	Red	5V supply	Pin 5V GPS
GND	Ground	Ground Signal	Pin GND GPS
TX2	Yellow	-	-
RX2	Black	Echo Sounder	DST800 Pin R(-) (1)
D8	Green	Echo Sounder Power Control	Relay Input Pin 3
D9	Red	Doppler Radar Power Control	Relay Input Pin 4
PWR Socket	Black	11 V Power	4 way splitter cable
USB TB	Gray	USB Serial Signal (Port MEGA2)	Raspberry Pi Serial Port

Note 1: as the RS422 to RS232 converter was not available, the RX2 line had been connected directly to the R(-) line. This allows communications, with some faulty messages in between.

Table 9 Teensy 3.1 A Pin Out.

Teensy 3.1 A Pin Out			
Pin Name	**Cable Color**	**Signal**	**To/ From**
SCL	Yellow	IMU I2C	IMU Mezzanine Pin 3
SDA	Red	IMU I2C	IMU Mezzanine Pin 4 – see Note 1
USB TμB	Black - Yellow	USB Serial Signal (Port TeensyA) and Power	Raspberry Pi Extender Board USB Port

Table 10 Teensy 3.1 B Pin Out

Teensy 3.1 B Pin Out			
Pin Name	**Cable Color**	**Signal**	**To/ From**
D5	Green	Doppler Radar fd Out	DB25F – WT26 - FPCDU
USB TμB	Black – Yellow	USB Serial Signal (Port TeensyB) and Power	Raspberry Pi Extender Board USB Port

Table 11 Teensy 3.1 C Pin Out

Teensy 3.1 C Pin Out			
Pin Name		**Signal**	**To/ From**
D1	Red	SA Distance (Maxsonar arrays)	Mega1 Pin 10
D2	Green	SA Depth (pressure sensor)	Mega1 Pin 11
D5	White	Roboteq Controller Power Control	Relay Input Pin 7 and 8
D6	Yellow	SA Switch IR	DB25F – WP24 – FPCDU Pin 48
PWM0	Red	Thruster 2 PWM	DB9 – WP8B – Port Thruster ESC
PWM1	Orange	Thruster 3 PWM	DB9 – WP8B – Center Thruster ESC
PWM2	White	Starboard Land Motor PWM	Roboteq Motor Controller 1 DB15 Pin 8 (2)
PWM3	Blue	Port Land Motor PWM	Roboteq Motor Controller 2 DB15 Pin 8 (2)
PWM4	Orange	Servo 1 PWM	DB9 – WP8B
PWM5	Green	Servo 2 PWM	DB9 – WP8B
PWM6	White	Thruster 1 PWM	DB9 – WP8B – Starboard Thruster ESC

Note 2: DB15 of Roboteq MC corresponds to PWM signal – 2nd priority. The first priority has been wired into a connector inside the cylinder, so if a remote control is connected, both MC will automatically follow the remote control commands.

Table 12 Motor Controller 1 Pin Out

Motor Controller 1 Pin Out			
Pin Name	**Cable Color**	**Signal**	**Remarks**
PWR	Red	22 V to MC	
GND	Black	Ground	
MOT1(+)	Red	Port Motor (+)	Both cables are merge together
MOT1(-)		Port Motor (+)	
MOT2(+)	Orange	Port Motor (-)	Both cables are merge together
MOT2(-)		Port Motor (-)	
DB15 Pin 2	Red	RS232 TX Out	To offline MC1MC2 connector
DB15 Pin 3	Yellow	RS232 RX In	To offline MC1MC2 connector
DB15 Pin 4	Blue	PWM Control 1	To offline MC1MC2 connector
DB15 Pin 5	Black	Ground	
DB15 Pin 8	White	PWM Control 2	
DB15 Pin 13	Black	Ground	To offline MC1MC2 connector

Table 13 Motor Controller 2 Pin Out

Motor Controller 2 Pin Out			
Pin Name	**Cable Color**	**Signal**	**Remarks**
PWR	Red	22 V to MC	
GND	Black	Ground	
MOT1(+)	Red	Starboard Motor (+)	Both cables are merge together
MOT1(-)		Starboard Motor (+)	
MOT2(+)	Orange	Starboard Motor (-)	Both cables are merge together
MOT2(-)		Starboard Motor (-)	
DB15 Pin 2	Red	RS232 TX Out	To offline MC1MC2 connector
DB15 Pin 3	Yellow	RS232 RX In	To offline MC1MC2 connector
DB15 Pin 4	White	PWM Control 1	To offline MC1MC2 connector
DB15 Pin 5	Black	Ground	
DB15 Pin 8	Blue	PWM Control 2	
DB15 Pin 13	Black	Ground	To offline MC1MC2 connector

Table 14 FWD - Forward Data Cable Connection

Signal	**Microprocessor**	**DB25 FWD**	**WT 24 Pin Connector**	**FPCDU**	**Remarks**
Doppler 11V	Relay Unit #4	1	1	1	To be converted to 5V in FPCDU.
IMU/ IRSIC 11V	Relay Unit #6	4	4	4	To be converted to 5V and 3.3V in FPCDU.
Doppler signal	FPCDU	6	6	6	Pulse train from the Doppler radar.
Ground	GND	7	7	7	
IMU SCL	Teensy 3.1A A5	9	9		
IMU SDA	Teensy 3.1A A4	10	10		
Analog 5	Mega1 A4	12	12	12	Analog signals from the Maxsonar forward array.
Analog 3	Mega1 A2	13	13	13	
Analog 1	Mega1 A0	14	14	14	
Analog 2	Mega1 A1	15	15	15	
Analog 4	Mega1 A3	16	16	16	
Tx Control	Mega1 D8	17	17	17	Used to trigger the Maxsonar array.
IR Switch SA	Teensy 3.1C D6	19	19	48	From IRSIC
Ground	GND	20	20	20	
FWD Maxsonar 11V	Relay Unit#1	21	21	21	To be converted to 5V in FPCDU.
Echo Sounder Rx	Mega2 Tx3	22	22		
Echo Sounder Tx	Mega2 Rx3	23	23		
GND	Ground	24	24	20	
Echo Sounder 11V	Relay Unit #3	25	2		

Table 15 AFT 1 and 2 – Aft Data Cable Connection

Signal	Microprocessor	DB25 AFT	WT8A Pin Connector	Cable Color	APCDU	Remarks
Analog 1	Mega1 A5	1	1	Gray	1	Analog signals form the Maxsonar aft array
Analog 2	Mega1 A6	2	2	White	2	
Analog 3	Mega1 A7	3	3	Red	3	
Aft Tx Control	Mega1 D9	4	4	Green	4	
Ground	GND	5, 11, 14				APCDU will be connected to ground through the IRSIC connection.
Pressure SDA	Mega1 SDA	7	5	Orange	7	
Pressure SCL	Mega1 SDL	8	6	Blue	8	
AFT Maxsonar 11V	Relay Unit#2	13	7	White Black	13	To be converted to 5V in APCDU.
Pressure 11V	Relay Unit#5	15	8	Red Black	15	To be converted to 3.3V in APCDU.

Table 16 PWM – PWM Data Cable Pin Out

Signal	DB9 AFT	Cable Color	WT8B Pin Connector	Cable Color
Thruster 1	1	White	1	Gray
Ground	2	Green	2	White
Thruster 2	3	Red	3	Red
Thruster 3	4	Orange	4	Green
11V Power	6	Blue	6	Blue
Servo 1	9	Orange	7	White – Black
Servo 2	8	Green	8	Red - Black

Table 17 12.5 Pair Cable Pin Out

Pin	Cable Color
1	Orange
2	Orange black
3	Gray
4	Gray black
5	Purple
6	Purple black
7	White
8	White black
9	Brown
10	Brown white
11	Blue
12	Blue white
13	Black
14	Light blue
15	Light blue black
16	Pink
17	Pink black
18	Red
19	Red black
20	Yellow
21	Yellow black
22	Light green
23	Light green black
24	Dark green
25	Dark green black

Note 3: this cable connects the FPCDU and APCDU thru the waterproof connectors.

Table 18 HVPS – Land Motors High Voltage Power Supply

Pin	Cable Color	Signal	Remarks
1	Red	MC1 Power	All cables goes through a power connector between WT12 and MCs.
2	Black	MC1 Ground	
3	Red	MC1 Mot1(+) Mot1(-)	
4			
5	Red	MC2 Power	
6	Orange	MC1 Mot2(+) Mot2(-)	
7			
8	Black	MC2 Ground	
10	Red	MC2 Mot1(+) Mot1(-)	
11			
9	Orange	MC2 Mot2(+) Mot2(-)	
12			

Table 19 LVPS – Electronics Low Voltage Power Supply

Pin	Cable Color (inside cylinder)	Cable Color (outside cylinder)	Signal	From	To
1	Red	Red	11 V in	Electronic switch	Electronics
2	Black	Black	Ground in	WT4 Pin 3	Electronics
3	Orange	Blue	Ground out	11 V battery (-)	WT4 Pin 2
4	Red	White	11 V out	11 battery (+)	Electronic switch

Note 4: inter-pin (3 and 4) connection is done in male connector outside cylinder. For battery recharge, WT4 exterior male connector is replaced, with only pins 3 and 4 connected.

Table 20 OFFLINE – Offline Connector

Pin	Cable Color	Signal	From	Remarks
1	Green	USB	Raspberry Pi2 USB Port	Cable to be used as an off line USB connection to Raspberry Pi and to Motor Controllers for set up.
2	White			
3	Red			
5	Black			
6	Black	Ground	MC2 DB15 Pin 13	
7	Brown	RS232 MC2 Rx in	MC2 DB15 Pin 2	
8	Red	RS232 MC2 Tx Out	MC2 DB15 Pin 3	
10	Black	Ground	MC1 DB15 Pin 13	
11	White	RS232 MC1 Rx in	MC1 DB15 Pin 2	
12	Green	RS232 MC1 Tx Out	MC1 DB15 Pin 3	

Table 21 ECHOIR – Echo Sounder and IRSIC

Signal	Cable Color	Molex 10 pin	WT9 Pin Connector	Cable Color	Remarks
Ground	Brown	1	1	Black	From WT9 connector cable will be separated
11 V	Orange	2	2	Red	
ECHO Tx	White	3	3	Blue	
Ground	Black	10	5	Black	From WT9 connector cable will be separated
5 V OP	Blue	9	6	Blue	
5 V SWIR	Red	8	7	Green	
IR Signal	Green	7	8	White	

Table 22 IR SWITCH – IR Switch Connector to IRSIC

Signal	Cable Color (sensor)	WT3 Pin Connector	Cable Color	Remarks
5 V	Red	1	Blue	It connects the IR Switches to the IRSIC module.
Ground	Green	2	Black	
Signal	Yellow	3	Green	

B. SOME BLOCK DIAGRAMS

Figure 71 shows a general block diagram of MOSARt. The interconnections represents the actual cabling, detailed in part A of the present Appendix.

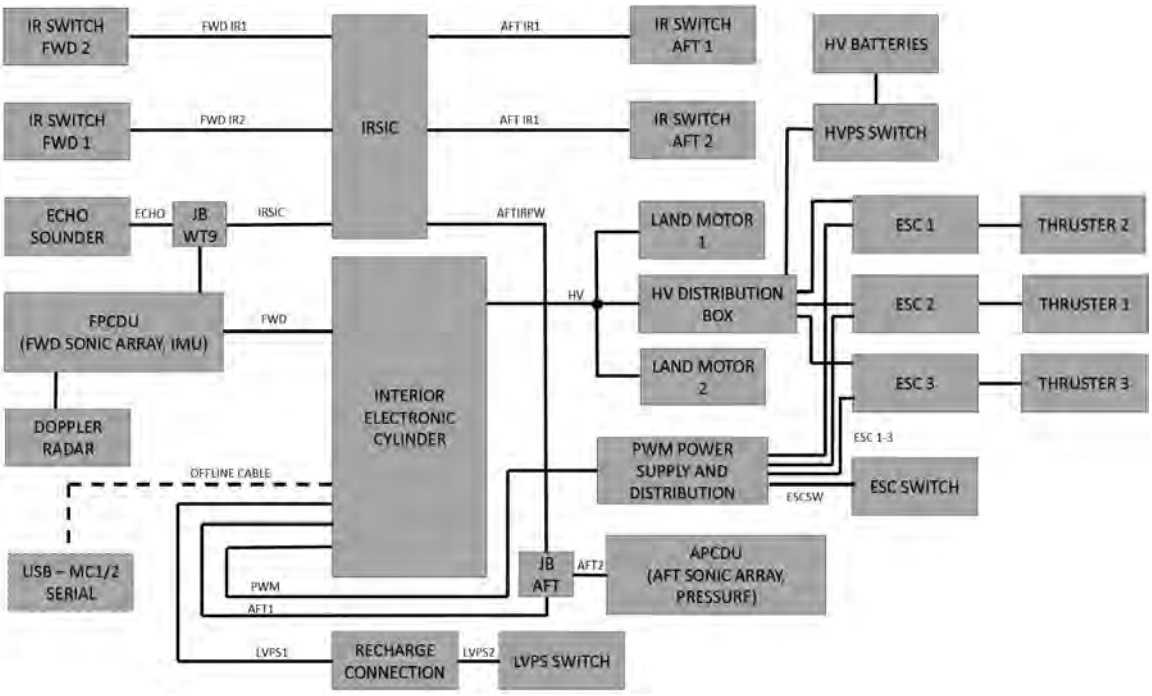


Figure 71 MOSARt Block Diagram

Figure 72 shows the block diagram and a photograph of the IRSIC board. This device allows to integrate the signals of the four IR Switch sensors.

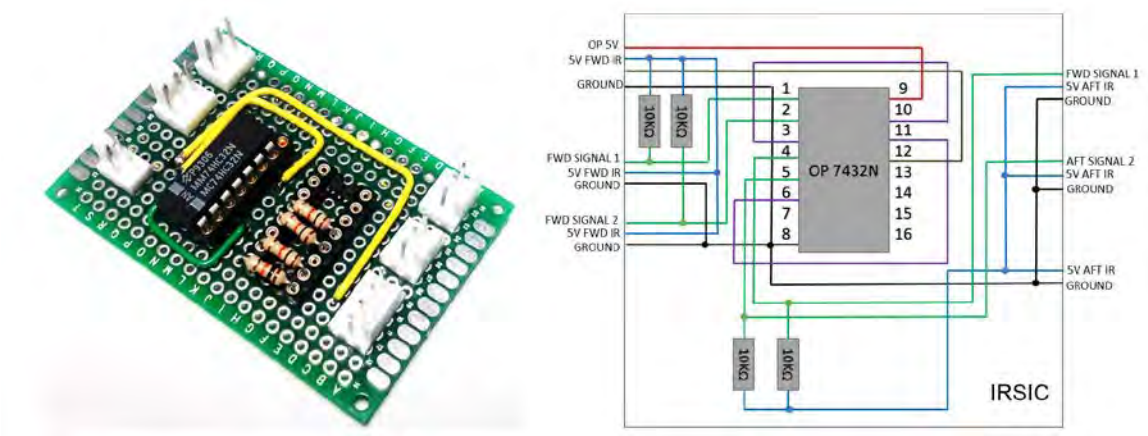


Figure 72 IRSIC Picture and Block Diagram

Figure 73 shows a general hardware inter connection. Color lines represents the different communication protocols and low power requirements of the devices.

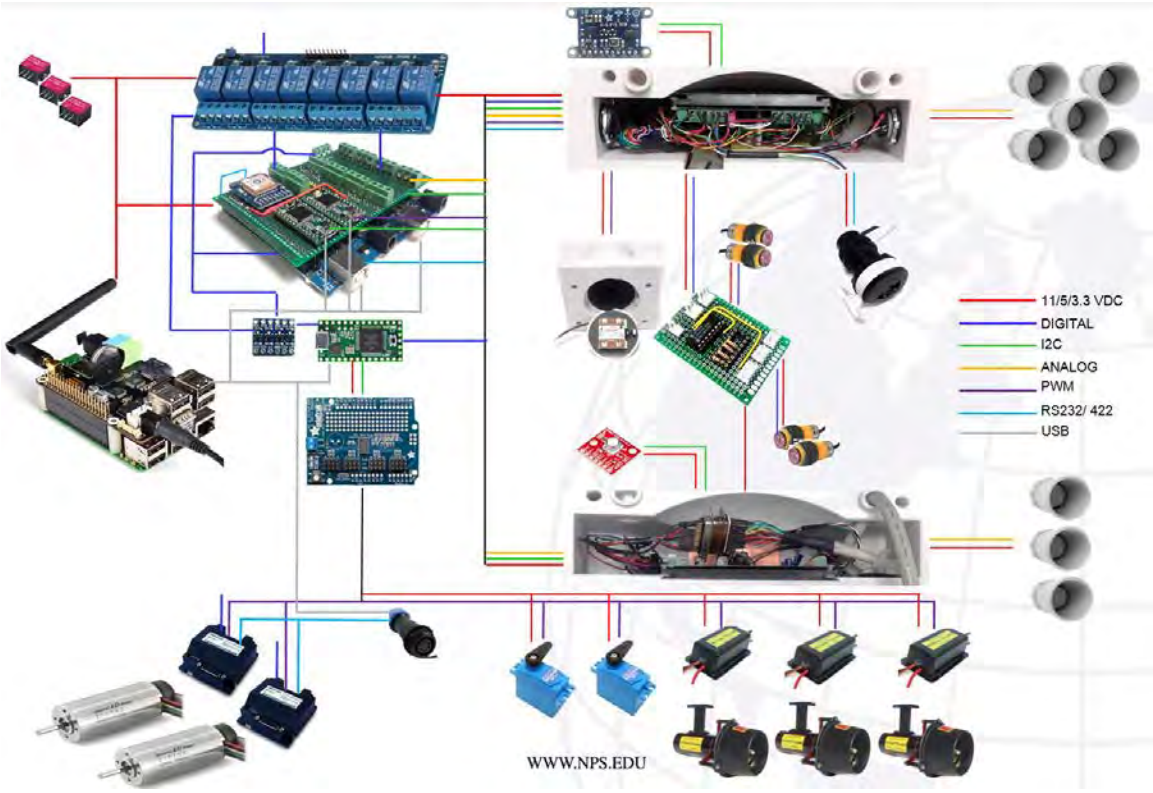


Figure 73 General Hardware Inter connection

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX C. SOFTWARE

A. MAIN PROCESSOR SOFTWARE

The program AXV_main.py integrates all the functions and allows MOSARt to act in an unmanned way. It is written on Python 3.4. For a better understanding, the related functions will be addressed first.

1. AXV_sensors.py

a. Functions

Table 23 Sensors Functions.

AXV_sensors.maxsonar		
Description	**Inputs**	**Outputs**
This function request information regarding the Maxsonar distance measurement, depth and temperature given by the pressure sensor board. The type of water to operate will trigger the calibration when it is called by the first time. The first call to forward or aft will power up each array, along with its associated IR switch array.	Mega1: name of serial port. Calwater: 0 or 1, indicates if MOSARt will operate in sweet water. Calseawater: 0 or 1, for sea water operation. SAdist: 0 or 1, deactivates/ activates SA signals for the Maxsonar sensors. SAdepth: same as above for the pressure sensor. Pres: 0 or 1, requests depth and temperature readings. Fwd: 0 or 1, request forward Maxsonar array. Aft: same as above for the aft array.	Cm1 to Cm5: integer, distance [cm] to closest obstacle detected by each Maxsonar sensor. If aft array is requested, Cm1 to Cm3 will hold the information and the rest will have dummy values. Depth: integer, depth [cm], calculated by the pressure sensor (positive down). Temp: [°C].

AXV_sensors.gps		
Description	**Inputs**	**Outputs**
Manages the power commands to the Doppler radar and the Echo sounder and request the latest GPS updates.	Mega2: name of serial port. Ecosonar: 0 or 1, will turn the power to the echo sounder on or off. Doppler: same as above, for the Doppler radar.	Timestamp: hour, minutes and seconds of the time that the GPS data was requested. Fixgps: 0 to 8, indicates the quality of the GPS reception (same as NMEA 183A). ErrorgpsRx: estimated fix error, in [m]. latRx, lonRx: float value. GPS's latitude and longitude information. AltgpsRx, velgpsRx, headgpsRx: altitude, velocitu and heading calculated by the GPS (float value) Goaldist: based on the fix information, it gives the estimated distance to the goal to achieve.
AXV_sensors.sonar		
Description	**Inputs**	**Outputs**
It request information regarding the echo sounder transducer.	Mega2: name of serial port	Depthson: float value. Echo sounder depth from the MOSARt to the ground sea, in [m]. Speedson: float value. Speed over ground calculated with the log. In [m/s].
AXV_sensors.IMU		
Description	**Inputs**	**Outputs**
It request heading, pitch and roll information.	TeensyA: name of serial port.	headRx: float value. Heading in degrees, ±180°. rollRx: float value, in degrees, ±90°. pitchRx: float value, in degrees, ±90°. Misc: dummy variable.

AXV_sensors.Doppler		
Description	**Inputs**	**Outputs**
It request velocity over ground measured by the Doppler radar.	TeensyB: name of serial port.	Veldopavg: float value, in [m/s]. Correspond to the average velocity measured from the last request. Veldop: float value, in [m/s]. Is the instantaneous velocity in the moment of request.

AXV_sensors. imumaxsonar		
Description	**Inputs**	**Outputs**
As the IMU presents the larger response delay, both request have been merged: while waiting for the response of the IMU, the program request information to the Maxsonar sub program.	Same as AXV_sensors.maxsonar and AXV_sensors.IMU	

b. Software

```
# AXV Sensors program
# Written by Oscar Garcia
# Physics department - Fall FY2016

import time

def maxsonar(Mega1, calwater, calseawater, SAdist, SAdepth, pres, fwd, aft):  # function to extract maxsonar array and barometric info
    cmdmega11 = calwater + 2*calseawater + 4*SAdist + 8*SAdepth # calibration is exclusive wr SA commands
    if(cmdmega11 == 0):
        cmdmega11 = 16 # 16 means no calibration, all SA deactivated
    cmdmega12 = pres + fwd*2 + aft*4
    flush = Mega1.read(Mega1.inWaiting())
    Mega1.write(bytes([90, cmdmega11, cmdmega12]))

    while(Mega1.inWaiting()<17):
            if(Mega1.inWaiting()>=18):
                break
    text1 = Mega1.read(Mega1.inWaiting())
    depth = (text1[14]*256 + text1[15])
    temp = (text1[16])
    cm1=text1[4]*256+text1[5]
    cm2=text1[6]*256+text1[7]
    cm3=text1[8]*256+text1[9]
    cm4=text1[10]*256+text1[11]
    cm5=text1[12]*256+text1[13]
    return(cm1, cm2, cm3, cm4, cm5, depth, temp)

def gps(Mega2, ecosonar, doppler):
    cmdmega21 = 2*ecosonar + doppler # doppler & sonar controls doppler rd on/ off (exclusive)
    if(cmdmega21 == 0):
        cmdmega21 = 4 # 4 means all off
    flush = Mega2.read(Mega2.inWaiting())
    Mega2.write(bytes([91, cmdmega21, 1]))

    while(Mega2.inWaiting()<19): # N-1 number of bytes of data from arduino
        if(Mega2.inWaiting()>=20):
            break
    text2 = Mega2.read(Mega2.inWaiting()) # will read the N number of data
    timestamp = str(time.strftime("%H%M%S"))
    fixgps = text2[2]
    errorgpsRx = float((text2[3]*256+text2[4])/100.0)
    latRx = int(((text2[5]*16777216)+(text2[6]*65536)+(text2[7]*256)+(text2[8])))
```

```
    if(latRx>2147483648):
        latRx = int(latRx - 4294967295)
    latRx = float(latRx/10000000.0)
    lonRx = int(((text2[9]*16777216)+(text2[10]*65536)+(text2[11]*256)+(text2[12])))
    if(lonRx>2147483648):
            lonRx = int(lonRx - 4294967295)
    lonRx = float(lonRx/10000000.0)
    altgpsRx = (text2[13]*256 + text2[14])
    if(altgpsRx>32768):
        altgpsRx = (int(altgpsRx - 65535))
    altgpsRx = float(altgpsRx/100.0)
    velgpsRx = float((text2[15]*256+text2[16])/100.0)*(1852.0/3600.0) # in knots
    headgpsRx = float(((text2[17]*256+text2[18]))/100.0)

    if(fixgps == 0):
        goaldist = 10
    elif(fixgps == 1):
        goaldist = 5
    elif(fixgps == 2):
        goaldist = 3
    else:
        goaldist = 5

    return(timestamp, fixgps, errorgpsRx, latRx, lonRx, altgpsRx, velgpsRx, headgpsRx, goaldist)

def sonar(Mega2): # on/off ecosounder & doppler is done with gps function
    flush = Mega2.read(Mega2.inWaiting())
    Mega2.write(bytes([92, 5, 2]))

    while(Mega2.inWaiting()<6):
        if(Mega2.inWaiting()>=7):
            break
    text3 = Mega2.read(Mega2.inWaiting())
    depthson = text3[2]*256+text3[3]
    speedson = (text3[4]*256+text3[5])/100.0  # PENDING
    #tempson = text3[6]

    return(depthson, speedson)

def IMU(TeensyA):
    TeensyA.write(bytes([93, 91, 200]))
    while(TeensyA.inWaiting()<13):
        if(TeensyA.inWaiting()>=13):
            break
    text4 = TeensyA.read(TeensyA.inWaiting())
    headRx = int(text4[4]*256+text4[5])
    if(headRx>32768):
        headRx = (int(headRx - 65535))
    headRx=headRx/100.0
    rollRx = int(text4[6]*256 + text4[7])
    if(rollRx>32768):
        rollRx = (int(rollRx - 65535))
    rollRx=rollRx/100.0
    pitchRx = int(text4[8]*256 + text4[9])
    if(pitchRx>32768):
        pitchRx = (int(pitchRx - 65535))
    pitchRx=pitchRx/100.0
    misc = (text4[10]*256+text4[11])/100.0 # available variable

    return(headRx, rollRx, pitchRx, misc)

def doppler(TeensyB):
    TeensyB.write(bytes([94, 91]))
    while(TeensyB.inWaiting()<3):
        if(TeensyB.inWaiting()>=4):
            break
    text5 = TeensyB.read(TeensyB.inWaiting())
    veldopavg = 2.0*float(text5[1]/100.0) # the x2 from trials
```

```
    veldop = 2.0*float(text5[2]/100.0)
    return(veldopavg, veldop)

def imumaxsonar(TeensyA, Mega1, calwater, calseawater, SAdist, SAdepth, pres, fwd, aft):

    TeensyA.write(bytes([93, 91, 200]))

    cmdmega11 = calwater + 2*calseawater + 4*SAdist + 8*SAdepth # calibration is exclusive wr SA commands
    if(cmdmega11 == 0):
        cmdmega11 = 16 # 16 means no calibration, all SA deactivated
    cmdmega12 = pres + fwd*2 + aft*4
    flush = Mega1.read(Mega1.inWaiting())
    Mega1.write(bytes([90, cmdmega11, cmdmega12]))

    while(Mega1.inWaiting()<17):
            if(Mega1.inWaiting()>=18):
                break
    text1 = Mega1.read(Mega1.inWaiting())
    depth = (text1[14]*256 + text1[15])
    temp = (text1[16])
    cm1=text1[4]*256+text1[5]
    cm2=text1[6]*256+text1[7]
    cm3=text1[8]*256+text1[9]
    cm4=text1[10]*256+text1[11]
    cm5=text1[12]*256+text1[13]

    while(TeensyA.inWaiting()<13):
        if(TeensyA.inWaiting()>=13):
            break
    text4 = TeensyA.read(TeensyA.inWaiting())
    headRx = int(text4[4]*256+text4[5])
    if(headRx>32768):
        headRx = (int(headRx - 65535))
    headRx=headRx/100.0
    rollRx = int(text4[6]*256 + text4[7])
    if(rollRx>32768):
        rollRx = (int(rollRx - 65535))
    rollRx=rollRx/100.0
    pitchRx = int(text4[8]*256 + text4[9])
    if(pitchRx>32768):
        pitchRx = (int(pitchRx - 65535))
    pitchRx=pitchRx/100.0
    misc = (text4[10]*256+text4[11])/100.0 # available variable

    return(headRx, rollRx, pitchRx, misc, cm1, cm2, cm3, cm4, cm5, depth, temp)
```

2. AXV_actuators.py

a. Functions

Table 24 Actuators Functions.

AXV_actuators.motorscontrol		
Description	**Inputs**	**Outputs**
It gives to the related microprocessor, the required information for perform PID control over the two land motors, three thrusters and 2 servos. It also manages SA actions directly from the Maxsonar sensors, the pressure depth calculation and the IR Switch array.	Due: name of serial port. head_error: is the relative head error with respect the calculated command heading. (+) indicates starboard side of MOSARt and (-) port side. depth_error: difference between the required and actual depth. servo_command: indicates the angle that the servo must adjust. Kland: 0 or 1, used to select conservative or aggressive parameters for PID Land control. Ksea: same as Kland for the thrusters. SA: indicates if the SA signals are going to be processed. Sea: 0 or 1, indicates if the information is for sea or land operation. Tail: 0 or 1, indicates to the preprocessor that the tail must follow the servo_command for the servo motors. Land: 0 or 1, indicates if MOSARt is over land. Direction: 0 or 1, indicates backward motion requested (0) or forward motion (1). Stop: 0 or 1, (0) means that MOSARt must head to the command heading. (1) indicates that MOSARt must stop. The heading of the message (first byte) can me modified to perform software reset (255), thruster calibraton (200) or motor controller calibration (210).	Output_land, Output_sea, Output_depth: it indicates the output of the corresponding PID. This information is for monitoring only. SAact: 0 or 1, indicates to the main program if the microprocessor has received a SA signal.

b. Software

```
# AXV Actuators program
# Written by Oscar Garcia
# Physics department - Fall FY2016

import struct

def motorscontrol(Due, head_error, depth_error, servo_command, Kland, Ksea, SA, sea, tail, land, direction, stop):
    [headlow,headhigh]=(struct.pack('<h', int(head_error)))
    [servolow,servohigh]=(struct.pack('<h', int(servo_command)))
    cmddue1 = land + 2*tail + 4*sea + 8*stop + 16*direction + 32*SA + Kland*64 + Ksea*128
    flush = Due.read(Due.inWaiting())
    Due.write(bytes([95, headlow, headhigh, servolow, servohigh, cmddue1, depth_error]))
    while(Due.inWaiting()<6):
        if(Due.inWaiting()>=7):
            break
    text5 = Due.read(Due.inWaiting())
    Output_land = int(text5[2])
    if(Output_land>128):
        Output_land = int(Output_land - 255)
    Output_sea = int(text5[3])
    if(Output_sea>128):
        Output_sea = int(Output_sea - 255)
    Output_depth = int(text5[4])
    if(Output_depth>128):
        Output_depth = int(Output_depth - 255)
    SAact = int(text5[5])

    # NOTE:
    # tail = 0 & + Servo_command = 512 will activate "turn in place"
    # Heading = 255 = reset PID
    # Heading = 200 = ESC calibration
    # Heading = 210 = MC calibration

    return(Output_land, Output_sea, Output_depth, SAact)
```

3. AXV_navigation.py

a. Functions

Table 25 Navigational Functions.

AXV_navigation.Haversine		
Description	**Inputs**	**Outputs**
It calculates the true bearing and distance between 2 latitude and longitude points, using the Haversine formula.	lat1: latitude of point 1. lat2: latitude of point 2. long1: longitude of point 1. long2: longitude of point 2.	Az: true azimuth (±180°) to point 2. D: distance [m] to point 2.
AXV_navigation.vpf		
Description	**Inputs**	**Outputs**
Using the information given by the Maxsonars, IMU and the position of the goal point, it calculates the heading error that will allow to reach the desired destination.	cm1, cm2, cm3, cm4, cm5: outputs of the Maxsonar sensors. az_true: true azimuth to goal. D: distance to goal. Head: current heading. Dirr: 0 or 1, 0 indicates going backwards, 1 going forward.	Totalmag: magnitude of the final heading vector. head_error: relative heading error (±180°).
AXV_navigation.drspeed		
Description	**Inputs**	**Outputs**
It calculates the true bearing, the relative heading error and distance to the point goal, using speed and IMU information.	Head: current heading. Pitch: average pitch from last calculation. Speed: speed of MOSARt. Deltat: time between calculations. az_true: true bearing to goal point. Dist: distance to goal point.	az_true: true bearing to goal point (±180°). Dist: distance tot goal point [m]. head_error: relative heading error to goal point (±180°).

AXV_navigation.drdist		
Description	**Inputs**	**Outputs**
It calculates the true bearing, the relative heading error and distance to the point goal, using calculated distance travelled and IMU information.	Head: current heading. Pitch: average pitch from last calculation. distavx: distance travelled by MOSARt since last calculation. az_true: true bearing to goal point. Dist: distance to goal point.	az_true: true bearing to goal point (±180°). Dist: distance tot goal point [m]. head_error: relative heading error to goal point (±180°).

AXV_navigation.kalmandop		
Description	**Inputs**	**Outputs**
It calculates the Kalman filtered velocity of MOSARt and the distance traveled between calculation.	Dopvel: MOSARt velocity extracted from the Doppler radar. Dt: time between calculations. Reset: 0 or 1, indicates reset parameters to initial conditions. Stop: 0 or 1, indicates that MOSARt has stop moving. A, H, Q, R: Kalman's filter system model variables. x: estimated variable for Kalman Filter. P: error covariance matrix.	Dist: filtered travelled distance [m]. Vel: filtered velocity [m/s]. A, H, Q, R, x, P: variables needed for next Kalman filter calculation. Reset: flag that indicates the state of the reset signal.

AXV_navigation.closesensor		
Description	**Inputs**	**Outputs**
It stipulates the actions to be taken when MOSARt detects objects that are less than 30 [cm] of distance.	cm1, cm2, cm3, cm4, cm5: outputs of the Maxsonar sensors. az_true: true bearing to point goal. Dist: distance to point goal. Head: heading of MOSARt. Dirr: 0 or 1, indicates the direction of MOSARt. 1 is	None.

	forward, 0 aft. Due: serial port for PID control. Mega1: serial port for Maxsonar sensors. TeensyA: serial port for IMU. Stop: 0 or 1. 1 indicates that MOSARt is in stop condition.	

b. Software

```
# AXV Navigation program
# Written by Oscar Garcia
# Physics department - Fall FY2016

def harversine(lat1, lat2, long1, long2):
    import math
    R = 6378137 # radius of earth in m
    lat1 = math.radians(lat1)
    lat2 = math.radians(lat2)
    long1 = math.radians(long1)
    long2 = math.radians(long2)
    d = 2*R*(math.asin(math.sqrt(math.pow((math.sin(0.5*(lat2-lat1))),2)+math.cos(lat1)*math.cos(lat2)*math.pow((math.sin(0.5*(long2-long1))),2))))
    az = math.degrees(math.atan2(math.sin(long2-long1)*math.cos(lat2), math.cos(lat1)*math.sin(lat2)-math.sin(lat1)*math.cos(lat2)*math.cos(long2-
long1)))
    # x = d*math.sin(math.radians(az))
    # y = d*math.cos(math.radians(az))
    return(az, d)#, x, y)


def vpf(cm1, cm2, cm3, cm4, cm5, az_true, d, head, dirr): # 18/10/15
    import math
    Repfactor = 4  # both factors tunned with matlab AXV_vff.m program
    Attfactor = 0.1
    Attthreshold = 10 # distance to change from conic and quadratic potential

    # repulsive potential

    if(dirr == 1):  # means forward motion
        if (cm1 >= 450):
            F1x = 0
            F1y = 0
        else:
            F1x = 0
            F1y = -100.0/(cm1)  # F = q/||q||^2
        if (cm2 >= 450):
            F2x = 0
            F2y = 0
        else:
            F2x = -100.0/(0.7071*cm2) # cos45 = 0.7071, x 100 to pass to m
            F2y = -100.0/(0.7071*cm2)
        if (cm3 >= 450):
            F3x = 0
            F3y = 0
        else:
            F3x = 100.0/(0.7071*cm3)
            F3y = -100.0/(0.7071*cm3)
        if (cm4 >= 150):
            F4x = 0
            F4y = 0
        else:
            F4x = -100.0/(cm4)
```

```
            F4y = 0
        if (cm5 >= 150):
            F5x = 0
            F5y = 0
        else:
            F5x = 100.0/(cm5)
            F5y = 0
    else: # means dirr == 0 ergo backwards motion
        if (cm1 >= 450):
            F1x = 0
            F1y = 0
        else:
            F1x = 0
            F1y = 100.0/(cm1)  # F = q/||q||^2
        if (cm2 >= 450):
            F2x = 0
            F2y = 0
        else:
            F2x = 100.0/(cm2*0.5) # sin30
            F2y = 100.0/(cm2*0.5) # sin45
        if (cm3 >= 450):
            F3x = 0
            F3y = 0
        else:
            F3x = -100.0/(cm2*0.5)
            F3y = 100.0/(cm2*0.5)
        F4x = 0
        F4y = 0
        F5x = 0
        F5y = 0
        d = 0 # backwards vpf is intended to just get away from obstacles.

    # atractive potential

    az_rel = az_true - head

    if(d <= Attthreshold): # quadratic potential
        Fax = d*math.sin(math.radians(az_rel))
        Fay = d*math.cos(math.radians(az_rel))
    else: # conic potential
        Fax = Attthreshold*d*math.sin(math.radians(az_rel))/(d)
        Fay = Attthreshold*d*math.cos(math.radians(az_rel))/(d)

    # total potential

    Ftx = Repfactor*(F1x + F2x + F3x + F4x + F5x) + Attfactor*Fax
    Fty = Repfactor*(F1y + F2y + F3y + F4y + F5y) + Attfactor*Fay

    Ftmag = math.sqrt(Ftx*Ftx + Fty*Fty)

    head_error = math.degrees(math.atan2(Ftx, Fty))

    return(Ftmag, head_error)


def drspeed(head, pitch, speed, deltat, az_true, dist):
    import math
    distavx = deltat*speed*math.cos(math.radians(pitch))
    x_wp = dist*math.sin(math.radians(az_true))
    y_wp = dist*math.cos(math.radians(az_true))
    x_avx = distavx*math.sin(math.radians(head))
    y_avx = distavx*math.cos(math.radians(head))
    deltax = x_wp - x_avx
    deltay = y_wp - y_avx
    dist = math.sqrt(deltax*deltax + deltay*deltay)
    az_true = math.degrees(math.atan2(deltax,deltay))
    head_error = head - az_true
    if(head_error > 180):
```

```
        head_error = head_error - 360
    elif(head_error < - 180):
        head_error = head_error + 360

    return(az_true, dist, head_error)


def drdist(head, pitch, distavx, az_true, dist):
    import math
    x_wp = dist*math.sin(math.radians(az_true))
    y_wp = dist*math.cos(math.radians(az_true))
    x_avx = distavx*math.sin(math.radians(head))
    y_avx = distavx*math.cos(math.radians(head))
    deltax = x_wp - x_avx
    deltay = y_wp - y_avx
    dist = math.sqrt(deltax*deltax + deltay*deltay)
    az_true = math.degrees(math.atan2(deltax,deltay))
    head_error = head - az_true
    if(head_error > 180):
        head_error = head_error - 360
    elif(head_error < - 180):
        head_error = head_error + 360

    return(az_true, dist, head_error)


def kalmandop(dopvel, dt, reset, stop, A, H, Q, R, x, P):
    import numpy as np
    import numpy.linalg as lina

    if((reset == 1) or (stop == 1)):
        #dt = 0.1
        #A = np.array([[1,dt],[0,1]])
        H = np.array([[0,1]])
        Q = np.array([[2,0],[0,1]]) #np.array([[0.0132,0],[0,0.0004]])   value used in bench tests
        R = 10  # 0.00054217 value used in bench tests
        x = np.array([[0,0.5]]).transpose() #np.array([[0,0.2]]).transpose() to define a initial velocity of 0.2 m/s
        P = np.array([[1,0],[0,1]])
        reset = 0

    A = np.array([[1,dt],[0,1]])
    xp = A.dot(x)
    Pp = (A.dot(P)).dot((A.transpose())) + Q

    K = (Pp.dot((H.transpose()))).dot((lina.inv((H.dot(Pp)).dot((H.transpose())) + R)))

    x = xp + K.dot((dopvel - H.dot(xp)))
    P = Pp - (K.dot(H)).dot(Pp)
    dist = x[0]
    vel = x[1]
    return(dist, vel, A, H, Q, R, x, P, reset)


def closesensor(cm1, cm2, cm3, cm4, cm5, az_true, dist, head, dirr, Due, Mega1, TeensyA, stop):
    import AXV_actuators
    import AXV_sensors
    import math
    import numpy as np
    import time
    [Totalmag, head_error] = vpf(cm1, cm2, cm3, cm4, cm5, az_true, dist, head, dirr)
    if(abs(head_error <= 90)):
        [Output_land, Output_sea, Output_depth, SAact] = AXV_actuators.motorscontrol(Due, head_error, 0, 512, 1, 0, 0, 0, 0, 1, dirr, stop) # tail = 0 &
+ Servo_command = 512 will activate "turn in place"
        while(abs(head_error > 5)): # here the AXV will turn on place until the heading error is small
            [head, roll, pitch, misc] = AXV_sensors.IMU(TeensyA)
            [cm1, cm2, cm3, cm4, cm5, depth, temp]= AXV_sensors.maxsonar(Mega1,1,0,0,0,0,dirr,int(not(dir)))
            [Output_land, Output_sea, Output_depth, SAact] = AXV_actuators.motorscontrol(Due, head_error, 0, 512, 1, 0, 0, 0, 0, 1, dirr, stop) # tail = 0
& + Servo_command = 512 will activate "turn in place"
```

```
    [Output_land, Output_sea, Output_depth, SAact] = AXV_actuators.motorscontrol(Due, head_error, 0, 0, 1, 0, 0, 0, 0, 1, dirr, stop) # now it will
just stop (once the heading error is small)
    stop = 0 # next PID call will go foward.
    [head, roll, pitch, misc] = AXV_sensors.IMU(TeensyA)
    [cm1, cm2, cm3, cm4, cm5, depth, temp]= AXV_sensors.maxsonar(Mega1,1,0,0,0,0,dirr,int(not(dir)))
    [Totalmag, head_error]= vpf(cm1, cm2, cm3, cm4, cm5, az_true, dist, head, dirr)
  else:
    [cm1, cm2, cm3, cm4, cm5, depth, temp]= AXV_sensors.maxsonar(Mega1,1,0,0,0,0,int(not(dir)),dirr) # activate aft sensors
    [head, roll, pitch, misc] = AXV_sensors.IMU(TeensyA)
    #head = np.array([[math.cos(math.pi),math.sin(math.pi)],[-math.sin(math.pi),math.cos(math.pi)]]) # rotation matrix (XY rotation in Z axis): to
rotate heading 180 deg
    if((head >= 0) and (head<=180)):
      head = head - 180
    else:
      head = head + 180
    [Totalmag, head_error] = vpf(cm1, cm2, cm3, cm4, cm5, az_true, dist, head, int(not(dir)))
    stop = 0 # next PID call will go aft
    t1 = time.time()
    deltat = 0
    SAact = 0
    while(deltat<5): # it will go aft away from objets (without considering target WP) for 5 seconds.
      if((cm1 < 35) or (cm2 < 35) or (cm3 < 35) or SAact == 1):
        [Output_land, Output_sea, Output_depth, SAact] = AXV_actuators.motorscontrol(Due, head_error, 0, 512, 1, 0, 0, 0, 0, 1, 0, 1) # tail = 0 &
+ Servo_command = 512 will activate "turn in place"
      else:
        [Output_land, Output_sea, Output_depth, SAact] = AXV_actuators.motorscontrol(Due, head_error,0, 0, 1, 0, 0, 0, 0, 1, 0, stop)
      [cm1, cm2, cm3, cm4, cm5, depth, temp]= AXV_sensors.maxsonar(Mega1,1,0,0,0,0,int(not(dir)),dirr) # activate aft sensors
      [head, roll, pitch, misc] = AXV_sensors.IMU(TeensyA)
      #head = np.array([[math.cos(math.pi),math.sin(math.pi)],[-math.sin(math.pi),math.cos(math.pi)]]) # rotation matrix (XY rotation in Z axis): to
rotate heading 180 deg
      if((head >= 0) and (head<=180)):
        head = head - 180
      else:
        head = head + 180
      deltat = time.time() - t1
    [Output_land, Output_sea, Output_depth, SAact] = AXV_actuators.motorscontrol(Due, head_error, 0, 0, 1, 0, 0, 0, 0, 1, dirr, 1) # stop and
activate foward mode
    time.sleep(3) # sufficient time to stop
    [cm1, cm2, cm3, cm4, cm5, depth, temp]= AXV_sensors.maxsonar(Mega1,1,0,0,0,0,dirr,int(not(dir))) # activate foward sensors
  return()
```

4. AXV_misc.py

a. Functions

Table 26 Miscellaneous Functions.

AXV_misc.initialsetup		
Description	**Inputs**	**Outputs**
Loads the information from a text file, regarding if SA is desired, MOSARt dimensions, type of water to operate.	None. It reads the file InitialSetup.txt	Activation flags for: calwater, calseawater, SAdist, SAdepth. Information in [m] of: length, wide, height, wheg radius, tail length

AXV_misc.landORsea		
Description	**Inputs**	**Outputs**
By monitoring the IMU data, it determines if MOSARt is on land or over the water surface.	TeensyA, Due: serial ports to be used. K1, K2, SA, sea, tail, land, direction, stop: flags for the PID control program.	Flags indicating activation of: sea, land, tail, stop, pres, fwd, aft, ecoson, doppler

AXV_misc.SerialSetup		
Description	**Inputs**	**Outputs**
It preforms the serial setup for the 5 communications ports.	None.	Mega1, Mega2, TeensyA, TeensyB, Due: name of the serial ports

AXV_misc.Waypoint		
Description	**Inputs**	**Outputs**
Loads the waypoints stored in a text file.	n: number of waypoint that information is required. Name of file: waypoint3.txt	A: numbers of waypoints. waypoint[n][0]: waypoint number waypoint[n][1]: latitude. waypoint[n][2]: longitude waypoint[n][3: type of waypoint (goal, climb, obstacle) waypoint[n][4]: last waypoint flag.

AXV_misc.printout		
Description	**Inputs**	**Outputs**
Function used to print out into the screen information regarding the sensors, status, etc.	Information to be printed: status, LoS, saltsweet, SAact, dirrec, stop, numwp, wpindex, az, dist, cm1, cm2, cm3, cm4, cm5, vel, head, roll, pitch, head_error, hour, lat, lon, rate	None. Just printout information into the screen.

AXV_misc.firstFix		
Description	**Inputs**	**Outputs**
It performs the first actions when trying to get the first GPS fix.	Mega2, direction, stop, land, sea, nums_wp, wp_id, wp_lat, wp_lon, wp_type, wp_flag	timestamp, fix, errorgps, lat, lon, altgps, velgps, headgps, goaldist, start_lat, start_lon, nums_wp, wp_id, wp_lat, wp_lon, wp_type, wp_flag

b. Software

```
# AXV Misc program
# Written by Oscar Garcia
# Physics department - Fall FY2016

import string
import array
import AXV_sensors
import AXV_actuators
import time
import os

def initialsetup():
    f = open('InitialSetup.txt').read()
    setup = [item.split() for item in f.split('\n')[:-1]]
    calwater = int(setup[1][0])
    calseawater = int(setup[1][1])
    SAdist = int(setup[1][2])
    SAdepth = int(setup[1][3])
    length = int(setup[1][4])
    wide = int(setup[1][5])
    height = int(setup[1][6])
    whegradius = int(setup[1][7])
    taillength = int(setup[1][8])
    return(calwater, calseawater, SAdist, SAdepth, length, wide, height, whegradius, taillength)

def col(B,j):
    Z=[ ]
    for i in range(len(B)):
        Z.append(B[i][j])
    return(Z)

def landORsea(TeensyA, Due, K1, K2, SA, sea, tail, land, direction, stop):
    import numpy as np
    import time
    [head1, roll1, pitch1, misc] = AXV_sensors.IMU(TeensyA)
    sea = 0
    tail = 0
    land = 0
    stop = 1
    (Due, 0, 0, 0, K1, K2, SA, 0, 0, land, direction, stop) # tells the PID controller to stop
    time.sleep(2)
    [head1, roll1, pitch1, misc] = AXV_sensors.IMU(TeensyA)
    time.sleep(1)
    [head2, roll2, pitch2, misc] = AXV_sensors.IMU(TeensyA)
    time.sleep(1)
    [head3, roll3, pitch3, misc] = AXV_sensors.IMU(TeensyA)
    time.sleep(1)
    [head4, roll4, pitch4, misc] = AXV_sensors.IMU(TeensyA)
    headarray = [head1, head2, head3, head4]
    pitcharray = [pitch1, pitch2, pitch3, pitch4]
    rollarray = [roll1, roll2, roll3, roll4]
    headstv = np.std(headarray)
    rollstv = np.std(rollarray)
    pitchstv = np.std(pitcharray)

    if((headstv >= 1.5)| (rollstv >= 1)|(pitchstv >= 1)):
        sea = 1
        land = 0
        pres = 1
        fwd = 0
        aft = 0
        ecoson = 1
        doppler = 0
    else:
        sea = 0
        land = 1
```

```
        pres = 0
        fwd = 1
        aft = 0
        ecoson = 0
        doppler = 1
    return(sea, land, tail, stop, pres, fwd, aft, ecoson, doppler)

def SerialSetup():
    import serial
    import time
    Mega1= serial.Serial(port='/dev/ttyUSB22',baudrate = 115200) # Create Mega1 for sonic, barometric
    Mega2= serial.Serial(port='/dev/ttyUSB21',baudrate = 115200) # Create Mega2 for GPS, sonar, doppler & sonar power control
    TeensyA= serial.Serial(port='/dev/ttyUSB23',baudrate = 115200) # Create TeensyA for IMU
    TeensyB= serial.Serial(port='/dev/ttyUSB24',baudrate = 115200) # Create TeensyB for Doppler
    Due= serial.Serial(port='/dev/ttyUSB25',baudrate = 115200) # Create Due for PID, SA (sonic, pressure, IR SW)

    Due.write(bytes([255, 4, 0, 0,0,0,8])) # reset the PID controller

    time.sleep(2)
    flush = Mega1.read(Mega1.inWaiting())
    flush = TeensyA.read(Mega1.inWaiting())
    flush = TeensyB.read(Mega1.inWaiting())
    flush = Due.read(Due.inWaiting())

    Mega2.write(bytes([91, 4, 1])) # first command to turn all off
    flush = Mega2.read(Mega2.inWaiting())

    [head, roll, pitch, misc] = AXV_sensors.IMU(TeensyA) # just to flush old data
    flush = TeensyA.read(Mega1.inWaiting())

    return(Mega1, Mega2, TeensyA, TeensyB, Due)

def Waypoint(n):
    wp = open('waypoint3.txt').read()
    waypoint = [item.split() for item in wp.split('\n')[:-1]] # N LAT LON TYPE LAST : these are the colums for the text file
    Num_wp = len(waypoint)
    a = 1
    while(a < Num_wp):
        waypoint[a][0]=int(waypoint[a][0])      # number of waypoint
        waypoint[a][1]=float(waypoint[a][1])    # latitude
        waypoint[a][2]=float(waypoint[a][2])    # longitude
        waypoint[a][3]=int(waypoint[a][3])      # type (goal, obstacle, climb)
        waypoint[a][4]=int(waypoint[a][4])      # flag that indicates if it is last wp
        a = a + 1
    a = a - 1 # number of loaded waypoints
    return(a, waypoint[n][0], waypoint[n][1], waypoint[n][2], waypoint[n][3], waypoint[n][4]) # outputs the total number of wp and requested waypoint
data

def printout(status, LoS, saltsweet, SAact, dirrec, stop, numwp, wpindex, az, dist, cm1, cm2, cm3, cm4, cm5, vel, head, roll, pitch, head_error, hour,
lat, lon, rate):
    os.system('clear')
    #os.system('cls')

    if (LoS == 1):
        LoS = 'Land'
    elif(LoS == 0):
        LoS = 'Sea'
    if (SAact == 1):
        SAact = 'On'
    elif(SAact == 0):
        SAact = 'Off'
    if (dirrec == 1):
        dirrec = 'FWD'
    else:
        dirrec = 'AFT'
    if(stop == 1):
        direcc = 'Stop'
    print('\nStatus: %s\n\nOn %s\t- SA: %s\t- Direc: %s\n\nRefresh: %d [Hz]\n'%(status, LoS, SAact, dirrec, rate))
```

```
    print('Num WP: %d\t- Current WP: %d\t- Az: %d [deg]\t- Dist %d[m]\nLast GPS: %s\t- Lat: %f\t- Lon: %f\n'%(numwp, wpindex, az, dist, hour, lat, lon))
    print('Sensors (left to right) [cm]:\t%d\t%d\t%d\t%d\t%d\n\nHead: %d [deg]\t- Roll: %d [deg]\t- Pitch: %d [deg]\t- Vel: %.2f [m/s]\nHead_error: %d [deg]'%(cm5, cm3, cm1, cm2, cm4, head, roll, pitch, vel, head_error))
    return()


def firstFix(Mega2, direction, stop, land, sea, nums_wp, wp_id, wp_lat, wp_lon, wp_type, wp_flag):
    import AXV_sensors
    import time
    fix = 0
    n=0
    while(fix == 0): # will loop until a valid gps fix is obtained
        printout('Waiting for valid GPS Fix...', land, 0, 0, direction, stop,nums_wp, wp_id, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,'?', 0, 0, 0)
        [timestamp, fix, errorgps, lat, lon, altgps, velgps, headgps, goaldist]=AXV_sensors.gps(Mega2, sea, land)
        start_lat = lat
        start_lon = lon
        n = n + 1
        if(n>=5):
            start_lat = wp_lat
            start_lon = wp_lon
            lat = wp_lat
            lon = wp_lon
            wp_id = wp_id + 1
            [nums_wp, wp_id, wp_lat, wp_lon, wp_type, wp_flag]= Waypoint(wp_id)
            fix =20
            timestamp = str(time.strftime("%H%M%S(WP)"))
            printout('No GPS valid Fix. Assuming WP1 as starting point', land, 0, 0, direction, stop,nums_wp, wp_id, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,0,0, timestamp, lat, lon, 0)
            time.sleep(2)
    if(fix < 10):
        printout('Valid GPS Fix obtained!!', land, 0, 0, direction, stop,nums_wp, wp_id, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,timestamp, lat, lon, 0)
        time.sleep(2)
    n = 0
    return(timestamp, fix, errorgps, lat, lon, altgps, velgps, headgps, goaldist, start_lat, start_lon, nums_wp, wp_id, wp_lat, wp_lon, wp_type, wp_flag)
```

5. AXV_climb.py

a. Functions

Table 27 Climb Functions.

AXV_climb.py		
Description	**Inputs**	**Outputs**
It performs the actions needed to climb a pre-defined obstacle. As the available servos do not allow to lift MOSARt weight, this function has the objective to be only a “proof of concept” and it must be run in a separate test platform. The processor related to the servo control uses the same software as the Teensy 3.1C of MOSARt.	None.	None.

b. Software

```
# AXV Climb program
# Written by Oscar Garcia
# Physics department - Fall FY2016

import struct
import AXV_sensors
import AXV_actuators
import time
import math
import serial

TeensyA= serial.Serial(port='com29',baudrate = 115200)
Due= serial.Serial(port='com16',baudrate = 115200)

print('Climbing test program for non AXV platform')
#name = str(time.strftime("%d%m%Y_%H%M%S_CLIMB.txt"))
name = input('Enter name of file to save:   \n')
print('Name of files to be saved: ')
print(name)
time.sleep(3)
f = open(name,'a')


def main():
    beta = 90
    tail = 1
    Due.write(bytes([255, 4, 0, 0,0,0,8])) # reset the PID controller
    flush = TeensyA.read(TeensyA.inWaiting())
    t1 = time.time()

    while(1):
        [head, roll, pitch, misc] = AXV_sensors.IMU(TeensyA)
        deltat = (time.time() - t1)
        if(pitch>5):
            beta = beta+3
        if(pitch<-10):
            beta = 90

        print('Time: %.3f [s]\tPitch: %.2f [deg]\tBeta: %d [deg]'%(deltat, pitch, beta))

        [servolow,servohigh]=(struct.pack('<h', int(beta)))
        cmddue1 = 1 + 2*tail + 4*0 + 8*0 + 16*1 + 32*0 + 0*64 + 0*128
        flush = Due.read(Due.inWaiting())
        Due.write(bytes([95, 0, 0, servolow, servohigh, cmddue1, 0]))
        f.write('\n%.3f\t%.2f\t%d'%(deltat, pitch, beta))
        #time.sleep(0.05)

if __name__ == '__main__':
    try:
        main()
    except KeyboardInterrupt:
        import serial
        import time
        print ('\n\n****Program Stopped****\n\n')
        Due.close()
        TeensyA.close()
        f.close()
```

6. AXV_main.py

a. Functions

The only function that is held in the main program is the program-interrupt routine, where it resets the sensors, commands the motors to stop and closes the serial ports and log files.

b. Software

```
# AXV Main program
# Written by Oscar Garcia
# Physics department - Fall FY2016


def main():
   import serial
   import time
   import os
   import struct
   import binascii
   import AXV_sensors
   import AXV_actuators
   import AXV_misc
   import AXV_navigation
   global Due
   global Mega1
   global Mega2
   global g
   global f

   # Serial port configuration
   AXV_misc.printout('Serial Configuration', '?', 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,'?', 0, 0, 0)
   [Mega1, Mega2, TeensyA, TeensyB, Due] = AXV_misc.SerialSetup()

   # Initial parameters loading
   AXV_misc.printout('Loading Initial Parameters', '?', 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,'?', 0, 0, 0)
   [calwater, calseawater, SAdist, SAdepth, length, wide, height, whegradius, taillength] = AXV_misc.initialsetup()
   time.sleep(1)
   AXV_misc.printout('Initial Paramenters Loaded', '?', 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,'?', 0, 0, 0)
   time.sleep(1)
   SA = SAdist & SAdepth
   direction = 1

   # Land or Sea operational enviroment determination
   AXV_misc.printout('Land or Sea Determination', '?', 0, 0, direction, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,'?', 0, 0, 0)
   [sea, land, tail, stop, pres, fwd, aft, ecoson, doppler] = AXV_misc.landORsea(TeensyA, Due, 0, 0, SA, 1, 0, 1, direction, 1)

   [timestamp, fix, errorgps, lat, lon, altgps, velgps, headgps, goaldist] = AXV_sensors.gps(Mega2, ecoson, doppler) # For turning on/ off
echosounder & Doppler

   # Sea enviroment set up.
   if(sea == 1):
      AXV_misc.printout('Sonic Sensors & Doppler OFF, Pressure & Ecosounder ON', land, 0, 0, direction, stop, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0,'?', 0, 0, 0)
      time.sleep(1)
      [cm1, cm2, cm3, cm4, cm5, depth, temp]= AXV_sensors.maxsonar(Mega1,calwater,calseawater,SAdist,SAdepth,pres,fwd,aft) # calibrate sweet
water
      if(calwater == 1):
         AXV_misc.printout('Sweet water calibration done!', land, 0, 0, direction, stop, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,'?', 0, 0, 0)
         time.sleep(1)
      if(calseawater == 1):
```

```
        AXV_misc.printout('Sea water calibration done!', land, 0, 0, direction, stop, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,'?', 0, 0, 0)
        time.sleep(1)
    [calwater,calseawater]=[0,0] # after calibration, these values are set to zero to avoid interference with other Mega1 functions
  if(land == 1):
    AXV_misc.printout('Sonic Sensors, Pressure & Doppler ON, Ecosounder OFF', land, 0, 0, direction, stop, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,'?',
0, 0, 0)
    time.sleep(1)

  # Just sets up the PID parameters to conservative Parameters.
  K1 = 1
  K2 = 1
  direction = fwd # if fwd =0, then it will be aft

  # Sending initial parameters to PID control. Stop = 1, so vehicle will not move
  [Output_land, Output_sea, Output_depth, SAact] = AXV_actuators.motorscontrol(Due, 0, 0, 10, K1, K2, SA, sea, tail, land, direction, stop)
  AXV_misc.printout('Initial Parameters sent to PID', land, 0, 0, direction, stop, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,'?', 0, 0, 0)
  time.sleep(1)

  # Files to be saved setup
  name1 = str(time.strftime("%d%m%Y_%H%M%S_sensors.txt"))
  name2 = str(time.strftime("%d%m%Y_%H%M%S_traking.txt"))
  print('\n ****** Name of files to be saved:\n\t%s (sensors)\n\t%s (tracking)\n' %(name1, name2))
  f = open(name1,'a')
  g = open(name2,'a')
  f.write('\n timerec, cm1, cm2, cm3, cm4, cm5, depth, temp, head, roll, pitch, vpn_mag, head_error\n')
  g.write('\n timerec, fix, errorgps, lat, lon, altgps, velgps, headgps, head, head_error, az_true, dist, wp_id, vel\n')
  time.sleep(2)

  # Loading the first waypoint
  [nums_wp, wp_id, wp_lat, wp_lon, wp_type, wp_flag]= AXV_misc.Waypoint(1) # loads the first waypoint
  AXV_misc.printout('Wayponts Loaded', land, 0, 0, direction, stop,nums_wp, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,'?', 0, 0, 0)
  time.sleep(1)

  [timestamp, fix, errorgps, lat, lon, altgps, velgps, headgps, goaldist, start_lat, start_lon, nums_wp, wp_id, wp_lat, wp_lon, wp_type, wp_flag] =
AXV_misc.firstFix(Mega2, direction, stop, land, sea, nums_wp, wp_id, wp_lat, wp_lon, wp_type, wp_flag)

  stop = 0 # nextime that PID is invoqued, AXV will move
  [head, roll, pitch, misc] = AXV_sensors.IMU(TeensyA)
  [az_true, dist] = AXV_navigation.harversine(start_lat, wp_lat, start_lon, wp_lon)
  head_error = 0 # dummy value

  AXV_misc.printout('Course to Waypoint calculated!', land, 0, 0, direction, stop,nums_wp, wp_id, az_true, dist, 0, 0, 0, 0, 0, 0, head, roll, pitch,
head_error, timestamp, lat, lon, 0)
  g.write('\n%s\t%d\t%.2f\t%.8f\t%.8f\t%.2f\t%.2f\t%.2f\t%.2f\t%.2f\t%.2f\t%.2f\t%d'%(timestamp, fix, errorgps, lat, lon, altgps, velgps, headgps,
head, head_error, az_true, dist, wp_id))
  time.sleep(1)

  if(sea == 1):
    wasatsea = 1
    depth_nav = 120 # required depth navigation in [cm]
    while(depth <= 20): # It will not activate central thruster till is 20 [cm] under water
      [timestamp, fix, errorgps, lat, lon, altgps, velgps, headgps, goaldist]=AXV_sensors.gps(Mega2, 1, 0)
      [head, roll, pitch, misc] = AXV_sensors.IMU(TeensyA)
      [cm1, cm2, cm3, cm4, cm5, depth, temp]= AXV_sensors.maxsonar(Mega1,calwater,calseawater,SAdist,SAdepth,pres,0,0)
      if(fix > 0):
        [az_true, dist] = AXV_navigation.harversine(lat, wp_lat, lon, wp_lon)
        [vpn_mag, head_error]= AXV_navigation.vpf(cm1, cm2, cm3, cm4, cm5, az, dist, head, dirr)
        g.write('\n%s\t%d\t%.2f\t%.8f\t%.8f\t%.2f\t%.2f\t%.2f\t%.2f\t%.2f\t%.2f\t%.2f\t%d'%(timestamp, fix, errorgps, lat, lon, altgps, velgps,
headgps, head, head_error, az_true, dist, wp_id))
        AXV_misc.printout('Waiting for manual inmersion. Updating course to next Waipoint...', land, 0, 0, direction, stop,nums_wp, wp_id, az_true,
dist, 0, 0, 0, 0, 0, 0, head, roll, pitch, head_error, timestamp, lat, lon, 0)
      else:
        AXV_misc.printout('No GPS fix while waiting for manual inmersion. AXV position is drifting!!!!', land, 0, 0, direction, stop,nums_wp, wp_id,
az_true, dist, 0, 0, 0, 0, 0, 0, head, roll, pitch, head_error, timestamp, lat, lon, 0)
    [head, roll, pitch, misc] = AXV_sensors.IMU(TeensyA)    # Now depth is ok to activate center thruster
    [az, dist, head_error] = AXV_navigation.drspeed(head, pitch, 0, 0, az, dist)        # head_error correction before PID (as manual inmersion could
change heading)
    [cm1, cm2, cm3, cm4, cm5, depth, temp]= AXV_sensors.maxsonar(Mega1,calwater,calseawater,SAdist,SAdepth,pres,fwd,aft)
```

```
        depth_error = depth - depth_nav # depth is given in [cm], positive value
        if(depth_error >= 255):
            depth_error = 255 # 255 [cm] is the max amount of error accepted by protocol
        [Output_land, Output_sea, Output_depth, SAact] = AXV_actuators.motorscontrol(Due, head_error, depth_error, pitch, K1, K2, SA, sea, tail, land, direction, stop) # AXV starts moving!
        [depthson, speedson, tempson]=AXV_sensors.sonar(Mega2)
        t1 = time.time()
        t2 = t1
        transition = 0
    else:
        wasatsea = 0

        while(sea == 1):
            time.sleep(1) # delay to give more time between calculations
            time = str(time.strftime("%H%M%S"))
            [cm1, cm2, cm3, cm4, cm5, depth, temp]= AXV_sensors.maxsonar(Mega1,calwater,calseawater,SAdist,SAdepth,pres,0,0)
            [head, roll, pitch, misc] = AXV_sensors.IMU(TeensyA)
            deltat = time.time() - t1 # for DR calculation
            [az_true, dist, head_error] = AXV_navigation.drspeed(head, pitch, speedson, deltat, az_true, dist)
            t1 = time.time()
            if(depth<= 20):
                depth_error = 0 # to avoid activating thruster without water (intake is 15 [cm] above depth sensor)
            else:
                depth_error = depth - depth_nav
            if(depth_error >= 255):
                depth_error = 255
            [Output_land, Output_sea, Output_depth, SAact] = AXV_actuators.motorscontrol(Due, head_error, depth_error, pitch, K1, K2, SA, sea, tail, land, direction, stop)
            deltat = time.time() - t2 # for timing ecosounder
            if(deltat > 5): # every 5 seconds sonar depth and speed is updated
                [depthson, speedson, tempson]=AXV_sensors.sonar(Mega2)
                t2 = time.time()
            if((depthson <= 0.5) and (depth <= depth_nav*1.1) and (dist <= 100)): # entering transition zone
                if(transition == 0): # only one adjustment per transition. If sonar depth increases, it will go out again
                    transition = 1
                    depth_nav = int(depth_nav*0.7) # it will decrease 30% the depth
                    t3 = time.time()
                deltat = time.time()-t3
                if((deltat >= 10) and (depth <= 20)): # after 10 s, if the depth is less than 20 cm, it will check if it is on land or sea
                    [Output_land, Output_sea, Output_depth, SAact] = AXV_actuators.motorscontrol(Due, head_error, depth_error, pitch, K1, K2, SA, sea, tail, land, direction, 1) # it will stop everything
                    time.sleep(2) # time needed to stop
                    [sea, land, tail, stop, pres, fwd, aft, ecoson, doppler] = AXV_misc.landORsea(TeensyA, Due, 0, 0, SA, 1, 0, 1, direction, 1) # will check land or sea
                    stop = 0
                    t3 = time.time()
            else:
                transition = 0
            g.write('\n%s\t%d\t%.2f\t%.8f\t%.8f\t%.2f\t%.2f\t%.2f\t%.2f\t%.2f\t%.2f\t%.2f\t%d'%(time, 20, 0, 0, depthson, depth, speedson, tempson, head, head_error, az_true, dist, wp_id))

    rate =0
    loopcounter = 0
    headavg = 0
    pitchavg = 0
    servo_command = 0

    if(land == 1):
        [A, H, Q, R, x, P, firstrun, depth_error] = [0, 0, 0, 0, 0, 0, 1, 0]
        t1 = time.time()
        t2 = t1
        if(wasatsea == 1): # means that came out of the water. Needs to clean up sensors!
            while((time.time()-t1)<=30): # 30 seconds of blind navigation (to get the water off the sensors!)
                [timestamp, fix, errorgps, lat, lon, altgps, velgps, headgps, goaldist]=AXV_sensors.gps(Mega2, 0, 1)
                [head, roll, pitch, misc] = AXV_sensors.IMU(TeensyA)
                if(fix > 0): # means that it has a fix!
                    [az_true, dist] = AXV_navigation.harversine(lat, wp_lat, lon, wp_lon)
                    [vpn_mag, head_error]= AXV_navigation.vpf(888, 888, 888, 888, 888, az_true, dist, head, fwd)
```

```
            else:
                time.sleep(1)
                deltat = time.time() - t1
                drspeed = 0.3 # as still there is water, doppler will not work efficiently
                t1 = time.time()
                [az_true, dist, head_error] = AXV_navigation.drspeed(head, pitch, drspeed, deltat, az_true, dist) # calculation with estimated speed (drspeed)
            [Output_land, Output_sea, Output_depth, SAact] = AXV_actuators.motorscontrol(Due, head_error, depth_error, pitch, K1, K2, SA, sea, tail, land, direction, stop)
            g.write('\n%s\t%d\t%.2f\t%.8f\t%.8f\t%.2f\t%.2f\t%.2f\t%.2f\t%.2f\t%.2f\t%d'%(timestamp, fix, errorgps, lat, lon, altgps, velgps, headgps, head, head_error, az_true, dist, wp_id))
            loopcounter = loopcounter + 1
            deltat = time.time() - t2
            if(deltat >= 10):
                rate = int((loopcounter+1)/10.0)
                loopcounter = 0
                t2 = time.time()
            AXV_misc.printout('Blind Navigation on Land! (with fix estimated speed)', land, 0, 0, direction, stop,nums_wp, wp_id, az_true, dist, cm1, cm2, cm3, cm4, cm5, drspeed, head, roll, pitch, head_error, timestamp, lat, lon, rate)

    gpstime = 10 # time in seconds for gps request
    loopcounter = 0
    status = 'Navigation on Land!'
    [veldopavg, veldop] = AXV_sensors.doppler(TeensyB) # for reseting the Teensy
    [Output_land, Output_sea, Output_depth, SAact] = AXV_actuators.motorscontrol(Due, head_error, depth_error, pitch, K1, K2, SA, sea, tail, land, direction, stop)
    n = 1
    while(wp_id <= (nums_wp+1)): # now is out of the blind zone
        [t1, t2] = [time.time(), time.time()]
        [timerec, fix, errorgps, newlat, newlon, altgps, velgps, headgps, goaldist]=AXV_sensors.gps(Mega2, 0, 1)
        if(fix > 0):
            [az_true, dist] = AXV_navigation.harversine(lat, wp_lat, lon, wp_lon)
            [lat, lon] = [newlat, newlon]
            timestamp = timerec
        while(dist>=goaldist):
            timerec = str(time.strftime("%H%M%S")) # time to record
            [head, roll, pitch, misc, cm1, cm2, cm3, cm4, cm5, depth, temp]= AXV_sensors.imumaxsonar(TeensyA, Mega1, 0, 0, SAdist, SAdepth, pres, fwd, aft) # faster processing
            headavg = headavg + head
            pitchavg = pitchavg + pitch
            n = n + 1
            if(n >= 20): # DR will be updated arround 1 seconds
                [veldopavg, veldop] = AXV_sensors.doppler(TeensyB)
                deltat = time.time() - t1
                t1 = time.time()
                [dopdist, vel, A, H, Q, R, x, P, firstrun] = AXV_navigation.kalmandop(veldopavg, deltat, firstrun, stop, A, H, Q, R, x, P)
                head = float(headavg/n)
                pitch = float(pitchavg/n)
                [az_true, dist, head_error] = AXV_navigation.drdist(headavg, pitchavg, dopdist, az_true, dist)
                fix = 0
                headavg = 0
                n = 0
                firstrun = 1
            deltat = time.time() - t2
            if(deltat >= gpstime): # GPS will be updated every 10 seconds
                [timerec, fix, errorgps, newlat, newlon, altgps, velgps, headgps, goaldist]=AXV_sensors.gps(Mega2, 0, 1)
                if(fix > 0):
                    lat = newlat
                    lon = newlon
                    timestamp = timerec
                    [az_true, dist] = AXV_navigation.harversine(lat, wp_lat, lon, wp_lon)
                    gpstime = 10
                else:
                    gpstime = 5
                t2 = time.time()
                rate = int((loopcounter+1.0)/deltat)
                loopcounter = 0
            [vpn_mag, head_error]= AXV_navigation.vpf(cm1, cm2, cm3, cm4, cm5, az_true, dist, head, fwd)
```

```
            if(((cm1 < 35) or (cm2 < 35) or (cm3 < 35)) and (abs(head_error)>= 90)):
                servo_command = 512
                status = 'Navigation on Land, turning on place!'
            else:
                if(abs(head_error) <= 30):
                    servo_command = 0
                    status = 'Navigation on Land!'
            [Output_land, Output_sea, Output_depth, SAact] = AXV_actuators.motorscontrol(Due, head_error, depth_error, servo_command, K1, K2,
SA, sea, tail, land, direction, stop)

            g.write('\n%s\t%d\t%.2f\t%.8f\t%.8f\t%.2f\t%.2f\t%.2f\t%.2f\t%.2f\t%.2f\t%.2f\t%d\t%.2f'%(timerec, fix, errorgps, lat, lon, altgps, velgps,
headgps, head, head_error, az_true, dist, wp_id, vel))
            f.write('\n%s\t%d\t%d\t%d\t%d\t%d\t%.1f\t%d\t%.2f\t%.2f\t%.2f\t%.3f\t%.2f'%(timerec, cm1, cm2, cm3, cm4, cm5, depth, temp, head, roll,
pitch, vpn_mag, head_error))
            AXV_misc.printout(status, land, 0, 0, direction, stop,nums_wp, wp_id, az_true, dist, cm1, cm2, cm3, cm4, cm5, vel, head, roll, pitch,
head_error, timestamp, lat, lon, rate)
            loopcounter = loopcounter + 1
        # out of the while loop: means that it arrived to wp
        wp_id = wp_id + 1 # arrived to wp. jump to next wp
        if(wp_id > nums_wp): # arrived to last wp. AXV will go to start position
            wp_lat = start_lat
            wp_lon = start_lon
            status = 'Last WP done! Heading to Start Position...'
        else:
            [nums_wp, wp_id, wp_lat, wp_lon, wp_type, wp_flag]= AXV_misc.Waypoint(wp_id) # loads the next waypoint
            wpid = wp_id

    head_error = 0 # start position arrived. Turn off all.
    stop = 1
    land = 0
    sea = 0
    g.close()
    f.close()
    AXV_misc.printout('AXV mission FINISHED!!!!', land, 0, 0, direction, stop,nums_wp, wpid, az_true, dist, cm1, cm2, cm3, cm4, cm5, 0, head, roll,
pitch, head_error, timestamp, lat, lon, rate)
    [cm1, cm2, cm3, cm4, cm5, depth, temp]= AXV_sensors.maxsonar(Mega1,0,0,0,0,1,0,0)
    [Mega1, Mega2, TeensyA, TeensyB, Due] = AXV_misc.SerialSetup() # this will turn all off
    Mega1.close()
    Mega2.close()
    TeensyA.close()
    TeensyB.close()
    Due.close()


if __name__ == '__main__':
  try:
    main()
  except KeyboardInterrupt:
    import serial
    import time
    import os
    import AXV_sensors
    import AXV_actuators
    import AXV_misc
    import AXV_navigation

    print ('\n\n****Program Stopped****\n\n')
    g.close()
    f.close()
    [cm1, cm2, cm3, cm4, cm5, depth, temp]= AXV_sensors.maxsonar(Mega1,0,0,0,0,1,0,0)
    [Mega1, Mega2, TeensyA, TeensyB, Due] = AXV_misc.SerialSetup() # this will turn all off
    time.sleep(2)
    Mega1.close()
    Mega2.close()
    TeensyA.close()
    TeensyB.close()
    Due.close()
```

7. AXV_functionTests.py

a. Functions

This program holds 15 options that allows to test different functionalities and perform calibration procedures. The instructions of each step are included in the program.

b. Software

```
# AXV function Tests  program
# Written by Oscar Garcia
# Physics department - Fall FY2016


import AXV_main
import AXV_sensors
import AXV_actuators
import AXV_navigation
import AXV_misc
import time
import os

def main():
   os.system('clear')
   [Mega1, Mega2, TeensyA, TeensyB, Due] = AXV_misc.SerialSetup()
   print('*****AXV test programs - Fall FY2016*****\n\n')
   print('1. 30 [s] PID control to correct heading to 000.\n2. 15 [s] run for velocity and distance DR measurements')
   print('3. Perpendicular heading to a wall\n4. Land or sea determination\n5. Time taken for each function')
   print('6. PID and SA\n7. Turn on place\n8. Climbing Obstacle\n9. Sea PID test (mantain 000 heading)\n10. Mantaining depth test\n11. Maxsonar and IMU tests\n12. Thrusters ESC calibration')
   print('13. Land Motor Controllers setup\n14. Magnetometer calibration\n15. Land Motor Controllers Calibration\n')

      selection = input('\nEnter test program to execute:\n')

   if(selection == '1'):
      depth_error = 0
      servo_command = 0
      Kland = int(input('Enter 1 for concervative K parameters, 0 for aggressive parameter\n'))
      Ksea =0
      SA = 0
      sea = 0
      tail = 0
      land = 1
      direction = 1
      stop = 0
      name = str(time.strftime("%d%m%Y_%H%M%S_PID.txt"))
      f = open(name,'a')
      f.write('\nDeltat\tHead_Error\tPitch\tRoll\tOutput_land\n')
      print('Name of file to save:\n\t%s'%(name))
      t1 = time.time()
      deltat = 0
      while(deltat<=30):
         [head, roll, pitch, misc] = AXV_sensors.IMU(TeensyA)
         head_error = head
         [Output_land, Output_sea, Output_depth, SAact] = AXV_actuators.motorscontrol(Due, head_error, depth_error, servo_command, Kland, Ksea, SA, sea, tail, land, direction, stop)
         deltat = time.time() - t1
         f.write('%.3f\t%.2f\t%.1f\t%.1f\t%.1f\n'%(deltat, head_error, pitch, roll, Output_land))
      stop = 1
      [Output_land, Output_sea, Output_depth, SAact] = AXV_actuators.motorscontrol(Due, head_error, depth_error, servo_command, Kland, Ksea, SA, sea, tail, land, direction, stop)
```

```
        print('Test progam finished!\n')
        f.close()

    if(selection == '2'):
        [A, H, Q, R, x, P, firstrun, depth_error] = [0, 0, 0, 0, 0, 0, 1, 0]
        servo_command = 0
        Kland = 1
        Ksea =0
        SA = 0
        sea = 0
        tail = 0
        land = 1
        direction = 1
        stop = 0
        name = str(time.strftime("raw20_%H%M%S_DR.txt")) #("20m_%d%m%Y_%H%M%S_DR.txt"))
        f = open(name,'a')
        print('Name of file to save:\n\t%s'%(name))
        deltat = 0
        [timerec, fix, errorgps, lat1, lon1, altgps, velgps, headgps, goaldist]=AXV_sensors.gps(Mega2, 0, 1)
        time.sleep(1)
        distdop = 0
        distraw=0
        [veldopavg, veldop] = AXV_sensors.doppler(TeensyB)
        t1 = time.time()
        t2 = t1
        print('go!')
        while(deltat<=45):

            [head, roll, pitch, misc] = AXV_sensors.IMU(TeensyA)
            head_error = head
            [Output_land, Output_sea, Output_depth, SAact] = AXV_actuators.motorscontrol(Due, head_error, depth_error, servo_command, Kland,
Ksea, SA, sea, tail, land, direction, stop)

            [veldopavg, veldop] = AXV_sensors.doppler(TeensyB)
            deltat1 = time.time() - t2
            t2 = time.time()

            [dopdist, vel, A, H, Q, R, x, P, firstrun] = AXV_navigation.kalmandop(veldopavg, deltat1, firstrun, stop, A, H, Q, R, x, P)
            distdop = vel*deltat1 + distdop
            distraw = veldopavg*deltat1 + distraw

            deltat = time.time() - t1
            f.write('%.3f\t%.3f\t%.3f\t%.3f\t%.3f\t%.3f\t%.3f\t%.3f\n'%(veldopavg, deltat1, deltat, veldop, dopdist, vel, head, roll, pitch,))
            print(deltat, deltat1, veldopavg, veldop, vel, dopdist, distdop, distraw)
            time.sleep(1)

        stop = 1
        [Output_land, Output_sea, Output_depth, SAact] = AXV_actuators.motorscontrol(Due, head_error, depth_error, servo_command, Kland, Ksea,
SA, sea, tail, land, direction, stop)
        time.sleep(3)
        [timerec, fix, errorgps, lat2, lon2, altgps, velgps, headgps, goaldist]=AXV_sensors.gps(Mega2, 0, 1)
        [az_true, dist] = AXV_navigation.harversine(lat1, lat2, lon1, lon2)
        f.write('Az: %.1f\tDist: %.2f [m] From GPS fix (last quality fix: %d)\n'%(az_true, dist, fix))
        print('Test progam finished!\n')
        f.close()

    if(selection == '3'):
        print('Place AXV facing a wall up to 45 [deg] off axis, arround 1 [m] distance\n')
        dummy = input('Press any key + enter when ready\n')
        name = str(time.strftime("%d%m%Y_%H%M%S_turn.txt"))
        f = open(name,'a')
        f.write('\nDeltat\tHead_Error\Output_land\n')
        print('Name of file to save:\n\t%s'%(name))
        t1 = time.time()
        [n, cm1avg, cm2avg, cm3avg, cm4avg, cm5avg, head_error] =  [0, 0, 0, 0, 0, 0, 100]
        [head, roll, pitch, misc] = AXV_sensors.IMU(TeensyA)
        while(head_error > 5):
            while(n<=20):
```

```
            [cm1, cm2, cm3, cm4, cm5, depth, temp]= AXV_sensors.maxsonar(Mega1,1,0,0,0,0,1,0)
            cm1avg = cm1avg + cm1
            cm2avg = cm2avg + cm2
            cm3avg = cm3avg + cm3
            cm4avg = cm4avg + cm4
            cm5avg = cm5avg + cm5
            n = n + 1
         cm1avg = float(cm1avg/n)
         cm2avg = float(cm2avg/n)
         cm3avg = float(cm3avg/n)
         cm4avg = float(cm4avg/n)
         cm5avg = float(cm5avg/n)

         [left, centerleft, center, centerright, right]=[cm5avg, cm3avg, cm1avg, cm2avg, cm4avg]
         sensorarray = [left, centerleft, center, centerright, right]
         mindist = sensorarray.index(min(sensorarray))
         if(mindist == 0):
            [x1, y1, x2, y2] = [-cm5avg,0,-0.707*cm3avg, 0.707*cm3avg]
         if(mindist == 1):
            if(center<=left):
               [x1, y1, x2, y2] = [-0.707*cm3avg, 0.707*cm3avg, 0, cm1avg]
            else:
               [x1, y1, x2, y2] = [-0.707*cm3avg, 0.707*cm3avg, -cm5avg,0,-0.707*cm3avg]
         if(mindist == 3):
            if(centerright<=centerleft):
               [x1, y1, x2, y2] = [0, cm1avg, 0.707*cm2avg, 0.707*cm2avg]
            else:
               [x1, y1, x2, y2] = [0, cm1avg, -0.707*cm3avg, 0.707*cm3avg]
         if(mindist == 4):
            if(right<=center):
               [x1, y1, x2, y2] = [0.707*cm2avg, 0.707*cm2avg, cm4avg, 0]
            else:
               [x1, y1, x2, y2] = [0.707*cm2avg, 0.707*cm2avg, 0, cm1avg]
         if(mindist == 5):
            [x1, y1, x2, y2] = [cm4avg,0, 0.707*cm2avg, 0.707*cm2avg]
         slope = float((x2-x1)/(y2-y1)) # to give the angle with respect y axis
         head_error = -math.degrees(math.atan2(slope))
         [Output_land, Output_sea, Output_depth, SAact] = AXV_actuators.motorscontrol(Due, head_error, 0, 512, 1, 0, 0, 0, 0, 1, 1, 0)
         deltat = time.time() - t1
         t1 = time.time()
         f.write('%.3f\t%.2f\t%.1f\t%.1f\t%.1f\t%.1f\n'%(deltat, head_error, Output_land))
      [Output_land, Output_sea, Output_depth, SAact] = AXV_actuators.motorscontrol(Due, head_error, 0, 0, 1, 0, 0, 0, 0, 1, 1, 1)
      print('Test progam finished!\n')
      f.close()

   if(selection == '4'):
      go = input('Land or Sea function test!\nPlace SZP1 on land or over water and press Enter')
      [sea, land, tail, stop, pres, fwd, aft, ecoson, doppler] = AXV_misc.landORsea(TeensyA, Due, 0, 0, 0, 1, 0, 1, 1, 1)
      print('sea = %d\nland= %d\ntail= %d\nstop= %d\npres= %d\nfwd= %d\n aft= %d\necoson%d\n= doppler= %d\n'%(sea, land, tail, stop, pres,
fwd, aft, ecoson, doppler))
      print('Test progam finished!\n')

   if(selection == '5'):
      print('Time delay per function measurement\n')
      time.sleep(1)
      t1 = time.time()
      [cm1, cm2, cm3, cm4, cm5, depth, temp]= AXV_sensors.maxsonar(Mega1,1,0,1,1,0,1,0)
      deltat = time.time() - t1
      print('sonic time: %f [ms]'%(1000*deltat))
      t1 = time.time()
      [head, roll, pitch, misc] = AXV_sensors.IMU(TeensyA)
      deltat = time.time() - t1
      print('IMU time: %f [ms]'%(1000*deltat))
      t1 = time.time()
      [timerec, fix, errorgps, newlat, newlon, altgps, velgps, headgps, goaldist]=AXV_sensors.gps(Mega2, 0, 1)
      deltat = time.time() - t1
      print('GPS time: %f [ms]'%(1000*deltat))
      t1 = time.time()
```

```
        [veldopavg, veldop] = AXV_sensors.doppler(TeensyB)
        deltat = time.time() - t1
        print('DOP time: %f [ms]'%(1000*deltat))
        t1 = time.time()
        [cm1, cm2, cm3, cm4, cm5, depth, temp]= AXV_sensors.maxsonar(Mega1,0,0,1,1,0,1,0)
        deltat = time.time() - t1
        print('sonic time: %f [ms]'%(1000*deltat))
        t1 = time.time()
        [head, roll, pitch, misc] = AXV_sensors.IMU(TeensyA)
        deltat = time.time() - t1
        print('IMU time: %f [ms]'%(1000*deltat))
        t1 = time.time()
        [timerec, fix, errorgps, newlat, newlon, altgps, velgps, headgps, goaldist]=AXV_sensors.gps(Mega2, 0, 1)
        deltat = time.time() - t1
        print('GPS time: %f [ms]'%(1000*deltat))
        t1 = time.time()
        [veldopavg, veldop] = AXV_sensors.doppler(TeensyB)
        deltat = time.time() - t1
        print('DOP time: %f [ms]'%(1000*deltat))
        t1 = time.time()
        [cm1, cm2, cm3, cm4, cm5, depth, temp]= AXV_sensors.maxsonar(Mega1,0,0,1,1,0,1,0)
        deltat = time.time() - t1
        print('sonic time (SA ON): %f [ms]'%(1000*deltat))
        t1 = time.time()
        [cm1, cm2, cm3, cm4, cm5, depth, temp]= AXV_sensors.maxsonar(Mega1,0,0,0,1,0,1,0)
        deltat = time.time() - t1
        print('sonic time (SA dist OFF): %f [ms]'%(1000*deltat))
        t1 = time.time()
        [cm1, cm2, cm3, cm4, cm5, depth, temp]= AXV_sensors.maxsonar(Mega1,0,0,1,0,0,1,0)
        deltat = time.time() - t1
        print('sonic time (SA depth OFF): %f [ms]'%(1000*deltat))
        t1 = time.time()
        [cm1, cm2, cm3, cm4, cm5, depth, temp]= AXV_sensors.maxsonar(Mega1,0,0,0,0,0,1,0)
        deltat = time.time() - t1
        print('sonic time (SA OFF - FWD): %f [ms]'%(1000*deltat))
        t1 = time.time()
        [cm1, cm2, cm3, cm4, cm5, depth, temp]= AXV_sensors.maxsonar(Mega1,0,0,0,0,0,0,1)
        deltat = time.time() - t1
        print('sonic time (SA OFF - TURN ON AFT): %f [ms]'%(1000*deltat))
        t1 = time.time()
        [cm1, cm2, cm3, cm4, cm5, depth, temp]= AXV_sensors.maxsonar(Mega1,0,0,0,0,0,0,1)
        deltat = time.time() - t1
        print('sonic time (SA OFF - AFT): %f [ms]'%(1000*deltat))
        t1 = time.time()
        [cm1, cm2, cm3, cm4, cm5, depth, temp]= AXV_sensors.maxsonar(Mega1,0,0,0,0,1,0,0)
        deltat = time.time() - t1
        print('sonic time (SA OFF - PRESS): %f [ms]'%(1000*deltat))
        [A, H, Q, R, x, P, firstrun, depth_error] = [0, 0, 0, 0, 0, 0, 1, 0]
        t1 = time.time()
        [dopdist, vel, A, H, Q, R, x, P, firstrun] = AXV_navigation.kalmandop(2, 0.4, firstrun, 0, A, H, Q, R, x, P)
        deltat = time.time() - t1
        print('kalman time (firstrun): %f [ms]'%(1000*deltat))
        t1 = time.time()
        [dopdist, vel, A, H, Q, R, x, P, firstrun] = AXV_navigation.kalmandop(2, 0.4, firstrun, 0, A, H, Q, R, x, P)
        deltat = time.time() - t1
        print('kalman time: %f [ms]'%(1000*deltat))
        print('Test progam finished!\n')

    if(selection == '6'):
        print('Land PID with SA actions test:  ')
        depth_error = 0
        servo_command = 0
        Kland = int(input('Enter 1 for concervative K parameters, 0 for aggressive parameter\n'))
        Ksea =0
        SA = 1
        sea = 0
        tail = 0
        land = 1
```

```
        direction = 1
        stop = 0
        name = str(time.strftime("%d%m%Y_%H%M%S_PIDSA.txt"))
        f = open(name,'a')
        f.write('\nDeltat\tHead_Error\tPitch\tRoll\tOutput_land\n')
        print('Name of file to save:\n\t%s'%(name))
        t1 = time.time()
        deltat = 0
        while(deltat<=30):
            [head, roll, pitch, misc] = AXV_sensors.IMU(TeensyA)
            head_error = head
            [Output_land, Output_sea, Output_depth, SAact] = AXV_actuators.motorscontrol(Due, head_error, depth_error, servo_command, Kland,
Ksea, SA, sea, tail, land, direction, stop)
            deltat = time.time() - t1
            f.write('%.3f\t%.2f\t%.1f\t%.1f\t%.1f\n'%(deltat, head_error, pitch, roll, Output_land))
        stop = 1
        [Output_land, Output_sea, Output_depth, SAact] = AXV_actuators.motorscontrol(Due, head_error, depth_error, servo_command, Kland, Ksea,
SA, sea, tail, land, direction, stop)
        print('Test progam finished!\n')
        f.close()

    if(selection == '7'):
        print('Turn-on-place test: Please place SZP1 off 000 heading')
        depth_error = 0
        servo_command = 512
        Kland = int(input('Enter 1 for concervative K parameters, 0 for aggressive parameter\n'))
        Ksea =0
        SA = 1
        sea = 0
        tail = 0
        land = 1
        direction = 1
        stop = 1
        name = str(time.strftime("%d%m%Y_%H%M%S_PIDSA.txt"))
        f = open(name,'a')
        f.write('\nDeltat\tHead_Error\tPitch\tRoll\tOutput_land\n')
        print('Name of file to save:\n\t%s'%(name))
        t1 = time.time()
        deltat = 0
        while(deltat<=30):
            [head, roll, pitch, misc] = AXV_sensors.IMU(TeensyA)
            head_error = head
            [Output_land, Output_sea, Output_depth, SAact] = AXV_actuators.motorscontrol(Due, head_error, depth_error, servo_command, Kland,
Ksea, SA, sea, tail, land, direction, stop)
            deltat = time.time() - t1
            f.write('%.3f\t%.2f\t%.1f\t%.1f\t%.1f\n'%(deltat, head_error, pitch, roll, Output_land))
        stop = 1
        [Output_land, Output_sea, Output_depth, SAact] = AXV_actuators.motorscontrol(Due, head_error, depth_error, servo_command, Kland, Ksea,
SA, sea, tail, land, direction, stop)
        print('Test progam finished!\n')
        f.close()

    if(selection == '8'):
        print('Test program intended for non SZP1 plataform. Run program AXV_Climb.py with test platform\n')
        time.sleep(4)
        [Output_land, Output_sea, Output_depth, SAact] = AXV_actuators.motorscontrol(Due, head_error, 0, 0, 1, 0, 0, 0, 0, 1, 1, 1)
        print('Test progam finished!\n')
        f.close()

    if(selection == '9'):
        depth_error = 0
        servo_command = 0
        Ksea = int(input('Enter 1 for concervative K parameters, 0 for aggressive parameter\n'))
        Kland =0
        SA = 0
        sea = 1
        tail = 0
        land = 0
```

```
        direction = 1
        stop = 0
        name = str(time.strftime("%d%m%Y_%H%M%S_PIDsea.txt"))
        f = open(name,'a')
        f.write('\nDeltat\tHead_Error\tPitch\tRoll\tOutput_sea\n')
        print('Name of file to save:\n\t%s'%(name))
        t1 = time.time()
        deltat = 0
        while(deltat<=30):
            [head, roll, pitch, misc] = AXV_sensors.IMU(TeensyA)
            head_error = head
            [Output_land, Output_sea, Output_depth, SAact] = AXV_actuators.motorscontrol(Due, head_error, depth_error, servo_command, Kland,
Ksea, SA, sea, tail, land, direction, stop)
            deltat = time.time() - t1
            f.write('%.3f\t%.2f\t%.1f\t%.1f\t%.1f\n'%(deltat, head_error, pitch, roll, Output_sea))
        stop = 1
        [Output_land, Output_sea, Output_depth, SAact] = AXV_actuators.motorscontrol(Due, head_error, depth_error, servo_command, Kland, Ksea,
SA, sea, tail, land, direction, stop)
        print('Test progam finished!\n')
        f.close()

    if(selection == '11'): # maxsonar and imu test: 30 seconds of measurement
        Array = '2'
        [calwater, calseawater, SAdist, SAdepth, pres]=[0,0,0,0,0]
        Array = int(input('Enter "1" for Foward Array, "0" for Aft Array\n'))
        comment = input('Insert Comment:\n')
        if(Array == 1):
            name = str("%s_FWD.txt"%comment)
        elif(Array == 0):
            name = str("%s_AFT.txt"%comment)
        elif(Array >1):
            Array =1
            print('Forward array selected by default')
        f = open(name,'a')
        print('Name of file to save:\n\t%s'%(name))
        t1 = time.time()
        deltat = 0
        while(deltat<=30):
            print(deltat)
            [head, roll, pitch, misc, cm1, cm2, cm3, cm4, cm5, depth, temp]=AXV_sensors.imumaxsonar(TeensyA, Mega1, calwater, calseawater,
SAdist, SAdepth, 0, Array, int(not(Array)))#Mega1, 1, 0, 1, 0, 0, Array, int(not(Array)))
            deltat = time.time() - t1
            f.write('%.3f\t%.2f\t%.1f\t%.1f\t%d\t%d\t%d\t%d\t%d\n'%(deltat, head, pitch, roll, cm1, cm2, cm3, cm4, cm5))
        print('Test progam finished!\n')
        f.close()

    if(selection == '12'):
        step1 = input('Thruster ESC Calibration.\n\n*** WARNING ***\nSwitch OFF SW2 (ESC Power Supply) before starting Test!!\nBe ready to turn
on when prompt. Press any key when ready.\n')
        Due.write(bytes([200, 4, 0, 0,0,0,8])) # calibration command to PID software
        time.sleep(1)
        print('TURN SW2 ON!!!')
        step2 = input('Excecuting Calibration... Press any key when ready.\n')
        print('Calibration finished!\n')

    if(selection == '13'):
        step1 = input('Connect OFFLINE connector and connect DB9 to a computer with running ROBOTEQ software. Press any key when ready.\n')
        [Output_land, Output_sea, Output_depth, SAact] = AXV_actuators.motorscontrol(Due, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 1) # turn on motor controllers
        step2 = input('Motors Controllers ON. Run ROBOTEQ software. Press any key when ready.\n')
        [Output_land, Output_sea, Output_depth, SAact] = AXV_actuators.motorscontrol(Due, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1)
        print('MCs OFF. Test progam finished!\n')

    if(selection == '14'):
        print('For calibration, you must upload the program TeensyMagCal into TeensyA and then run the python program AXV_magcal.py.')
        print('TeensyMagCal will output the magnetometers value on X Y Z. Rotate the vehicle on\nthe 3 axis until no new updates apears. Then press
Ctrl-C to upload calibration values into TeensyA EEPROM')

    if(selection == '15'):
```

```
    step1 = input('Connect OFFLINE connector and connect DB9 to a computer with running ROBOTEQ software.\n\n*** WARNING ***\nWhegs will go FULL forward and FULL aft!!')
    stept1 = input('Place MOZARt in a test bench to avoid movements.\n When calibration starts, Press OK to load parameters in ROBOTEQ software.\nThen go into Configuration - Inputs/Outputs - Pulse Inputs (second pulse) - Calibration.\nPress any key when ready.\n')
    Due.write(bytes([210, 4, 0, 0,0,0,17]))
    print('Running test. Expected duration: 60 s\n')
    time.sleep(60)
    print('MCs OFF. Test progam finished!\nTo use PID software again, restart Raspberry Pi')

if __name__ == '__main__':
  try:
    main()
  except KeyboardInterrupt:
    print ('\nProgram Stopped')
```

B. PREPROCESSORS SOFTWARE

Table 28 Maxsonars Forward/ Aft array and Pressure sensor

Pre Processor	Software Name
MEGA1	AXV_Mega1

```
#include <Wire.h>
#include <MS5803_I2C.h> // for pressure & temp sensor

MS5803 sensor(ADDRESS_HIGH); // adress high is 0x76 (default) for the pressure sensor


// ANALOG INPUTS
// Fowards sonic sensors
const int anDist1=0; // center
const int anDist2=1; // 45 right
const int anDist3=2; // 315 left
const int anDist4=3; // 90 right
const int anDist5=4; // 270 left
// Aft sonar sensors
const int anDist6=5; // 180 center
const int anDist7=6; // 135 right
const int anDist8=7; // 225 left

// PRESSURE & TEMP VARIABLES
double temperature_c, alt, pressure_abs, pressure_relative, pressure_baseline, pressure_absRaw, temperature_cRaw, temperature_cOld, pressure_absOld;
int sampleNum   = 20; // for the average filter calculation
float alpha     = 0.5; // coeficient for the 1st order lp
unsigned long t1 = 0;
```

```
int density; // water density [kg/m3]
//density = 1030; // density for salt water [kg/m3]

// CONTROL
// Status control
int FwdAftPre; // 0: Foward sensors, 1 Aft sensor, 2: Pressure & Temp.
int estatus; // AXV status
// Control signals to MaxSonar
const int TrigRXFwd   = 8; // PIN to RX for trigger first MaxSonar sensor of foward array
const int PWFwd       = 12; // enables power to the MaxSonar foward sensor array and FWD IR switches
const int TrigRXAft   = 9; // PIN to RX for trigger first MaxSonar sensor of aft array
const int PWAft       = 14; // enables power to the MaxSonar aft sensor array and AFT IR switches
const int SAdist      = 35; // distance that AXV will stop automatically
const int SAdPin      = 10; // signal to stop directly to PID Pre Processor
const int SAdepth     = 500; // depth that AXV will go up automatically
const int SAdefPin    = 11; // signal to stop directly to PID Pre Processor
boolean FWD           = false;
boolean AFT           = false;
boolean PresCal       = false;
const boolean ON      = LOW;
const boolean OFF     = HIGH;
boolean SA_dist = false; // by default SAs are deactivated
boolean SA_dept = false;

// DATA
// from MaxSonar sensors
int cm1, cm2, cm3, cm4, cm5, cm6, cm7, cm8;
float temp, TempComp; // Temperature sensor & temperature compensation
byte data[4]; // status dirr and heading from RPI
float RPIhead; // heading from RPI

void setup() {
   sensor.reset(); sensor.begin(); delay(1000); // pressure sensor setup  Serial.begin(115200);
   Serial.begin(115200); // serial port for comms to RPI
   pinMode(PWFwd, OUTPUT);
   pinMode(PWAft, OUTPUT);
   pinMode(TrigRXFwd, OUTPUT);
   pinMode(TrigRXAft, OUTPUT);
   pinMode(SAdPin, OUTPUT);
   pinMode(SAdefPin, OUTPUT);
   digitalWrite(PWFwd, OFF); // 06 Abri: new relay module uses invert logic
   digitalWrite(PWAft, OFF);
   TempComp=1;
}

void Pressure_Cal(){
  pressure_baseline = 0;
  for(int n = 0;n<20; n++)
  {
     pressure_baseline = pressure_baseline+sensor.getPressure(ADC_4096);
  }
  pressure_baseline = 100*pressure_baseline/20; // [Pa]
  PresCal = true;
  pressure_abs = 100*sensor.getPressure(ADC_4096);
  temperature_c = sensor.getTemperature(CELSIUS, ADC_4096);
```

```
  PresCal = true;
}

void Alt_Temp(){
  temperature_cOld= temperature_c;
  pressure_absOld = pressure_abs;
  temperature_cRaw = sensor.getTemperature(CELSIUS, ADC_4096); // [C]
  pressure_absRaw = 100*sensor.getPressure(ADC_4096); // [Pa]
  pressure_abs = alpha*pressure_absOld + (1-alpha)*pressure_absRaw;  // calulate the lpf data
  temperature_c = alpha*temperature_cOld + (1-alpha)*temperature_cRaw; // [deg C]
  alt = -100*(pressure_abs-pressure_baseline)/(density*9.81); // [cm]
  if((alt < -SAdepth)&&(SA_dept == true)) digitalWrite(SAdefPin, HIGH);
  else digitalWrite(SAdefPin, LOW);
}

void start_sensorFwd(){
  digitalWrite(PWAft, OFF); // turn off MaxSonar aft sensors
  pinMode(TrigRXFwd, OUTPUT); // first two steps is to ensure a start from off
  digitalWrite(PWFwd, ON); // turn on MaxSonar foward sensors
  delay(300); // minimum 250 ms is needed for sensors boot up.
  digitalWrite(TrigRXFwd,HIGH); // this will shoot the first sensor (RX of MaxSonar)
  delayMicroseconds(25); // minimum is 20us (for the trigger signal to take effect)
  digitalWrite(TrigRXFwd,LOW); // trigger signal applies only 1 to sensor 1 (then sensors will trigger each other in the
ring)
  pinMode(TrigRXFwd, INPUT); // triggerPin1 must now remain in high impedance so signal goes to rx of sensor 1..
  delay(600); // this is the relay required for calibration (only first readings).
  FWD = true;
  AFT = false;
}

void start_sensorAft(){
  digitalWrite(PWFwd, OFF); // turn off MaxSonar foward sensors
  pinMode(TrigRXAft, OUTPUT); // first two steps is to ensure a start from off
  digitalWrite(PWAft, ON); // turn on MaxSonar aft sensors
  delay(300); // minimum 250 ms is needed for sensors boot up.
  digitalWrite(TrigRXAft,HIGH); // this will shoot the first sensor (RX of MaxSonar)
  delayMicroseconds(25); // minimum is 20us (for the trigger signal to take effect)
  digitalWrite(TrigRXAft,LOW); // trigger signal applies only 1 to sensor 1 (then sensors will trigger each other in the
ring)
  pinMode(TrigRXAft, INPUT); // triggerPin1 must now remain in high impedance so signal goes to rx of sensor 1..
  delay(600); // this is the relay required for calibration (only first readings).
  FWD = false;
  AFT = true;
}

void Sensor_Fwd()
{
  cm1 = int(TempComp*analogRead(anDist1)*0.4961); // 254/512 conversion factor to cm
  cm2 = int(TempComp*analogRead(anDist2)*0.4961);
  cm3 = int(TempComp*analogRead(anDist3)*0.4961);
  cm4 = int(TempComp*analogRead(anDist4)*0.4961);
  cm5 = int(TempComp*analogRead(anDist5)*0.4961);
  if(cm1<=SAdist||cm2<=SAdist||cm3<=SAdist||cm4<=SAdist||cm5<=SAdist       &&       (SA_dist       ==       true))
digitalWrite(SAdPin, HIGH);
  else digitalWrite(SAdPin, LOW);
```

```
}

void Sensor_Aft()
{
  cm6 = int(TempComp*analogRead(anDist6)*0.4961);
  cm7 = int(TempComp*analogRead(anDist7)*0.4961);
  cm8 = int(TempComp*analogRead(anDist8)*0.4961);
  /*if(cm6<=SAdist||cm7<=SAdist||cm8<=SAdist && (SA_dist == true)) digitalWrite(SAdPin, HIGH);
  else digitalWrite(SAdPin, LOW);*/
}

void printFwdSen()
{
  Serial.write(90);
  Serial.write(91);  // ID MEGA 1
  Serial.write(highByte(301));
  Serial.write(lowByte(301)); // ID for foward sensors
  Serial.write(highByte(cm1));
  Serial.write(lowByte(cm1));
  Serial.write(highByte(cm2));
  Serial.write(lowByte(cm2));
  Serial.write(highByte(cm3));
  Serial.write(lowByte(cm3));
  Serial.write(highByte(cm4));
  Serial.write(lowByte(cm4));
  Serial.write(highByte(cm5));
  Serial.write(lowByte(cm5));
  Serial.write(highByte(int(alt)));
  Serial.write(lowByte(int(alt)));
  Serial.write(int(temperature_c));
  Serial.write(170);
}

void printAftSen()
{
  Serial.write(90);
  Serial.write(91);  // ID MEGA 1
  Serial.write(highByte(302));
  Serial.write(lowByte(302)); // ID for aft sensors
  Serial.write(highByte(cm6));
  Serial.write(lowByte(cm6));
  Serial.write(highByte(cm7));
  Serial.write(lowByte(cm7));
  Serial.write(highByte(cm8));
  Serial.write(lowByte(cm8));
  Serial.write(highByte(888));
  Serial.write(lowByte(888));
  Serial.write(highByte(888));
  Serial.write(lowByte(888));
  Serial.write(highByte(int(alt)));
  Serial.write(lowByte(int(alt)));
  Serial.write(int(temperature_c));
  Serial.write(170);
}
```

```
void printPresTemp()
{
  Serial.write(90);
  Serial.write(91);  // ID MEGA 1
  Serial.write(highByte(303));
  Serial.write(lowByte(303)); // ID for pressure & temperature only
  Serial.write(highByte(888));
  Serial.write(lowByte(888));
  Serial.write(highByte(888));
  Serial.write(lowByte(888));
  Serial.write(highByte(888));
  Serial.write(lowByte(888));
  Serial.write(highByte(888));
  Serial.write(lowByte(888));
  Serial.write(highByte(888));
  Serial.write(lowByte(888));
  Serial.write(highByte(int(alt)));
  Serial.write(lowByte(int(alt)));
  Serial.write(int(temperature_c));
  Serial.write(170);
}

void commsRPI(){
  while(Serial.available()<3){
    if (FWD == true) Sensor_Fwd();
    if (AFT == true) Sensor_Aft();
    if (PresCal == true) Alt_Temp();
  }
  data[0]=Serial.read(); // heading
  data[1]=Serial.read(); // 1: calibrate Pressure command WATER, 2: calibrate Pressure command SEA, 4: SA dist
activated, 8: SA dist deactivated, 16: SA depth activated, 32: SA depth deactivated
  data[2]=Serial.read(); // 1: Pres/Temp only; 2: FWD; 3: AFT; 2 or 3 + 1 Pres/Temp update

  switch (data[1]){
    case 1: // calibrate command for pure water
      density = 1000;
      Pressure_Cal(); printPresTemp();
      break;
    case 2: // calibrate command for salt water
      density = 1300;
      Pressure_Cal(); printPresTemp();
      break;
    case 4: // SA distance is activated
      SA_dist = true;
      break;
    case 8: // SA depth is activated
      SA_dept = true;
      break;
    case 16: // all SA deactivated
      SA_dist = false;
      SA_dept = false;
      break;
  }
  switch (data[2]){
```

```
    case 1:          // only pressure and temperature
     if(FWD == true || AFT == true){
       digitalWrite(PWFwd, OFF);
       FWD = false;
       digitalWrite(PWAft, OFF);
       AFT = false;
     }
     Alt_Temp(); printPresTemp();
     break;
    case 2:         // foward sensors, no pressure & temperature update
     if(FWD == false) start_sensorFwd();
     Sensor_Fwd(); printFwdSen();
     break;
    case 4:         // aft sensors, no pressure & temperature update
     if(AFT == false) start_sensorAft();
     Sensor_Aft(); printAftSen();
     break;
    case 3:         // foward sensors + pressure & temperature update
     if(FWD == false) start_sensorFwd();
     if(PresCal == false) Pressure_Cal();
     Alt_Temp(); Sensor_Fwd(); printFwdSen();
     break;
    case 5:         // aft sensors + pressure & temperature update
     if(AFT == false) start_sensorAft();
     if(PresCal == false) Pressure_Cal();
     Alt_Temp(); Sensor_Aft(); printAftSen();
     break;
  }  // switch data[2] bracket

} // commsPRI bracket

void loop() {
  commsRPI();
}
```

Table 29 GPS, DST800 Software.

Pre Processor	Software Name
MEGA2	AXV_Mega2

```
#include <Adafruit_GPS.h>    // Adafruit GPS library
#include <SoftwareSerial.h>

#define SONARSerial Serial2

Adafruit_GPS GPS(&Serial3);
HardwareSerial mySerial = Serial3;

char gpsdata; //to read characters coming from the GPS
byte data[4];
```

Pre Processor	Software Name
MEGA2	AXV_Mega2

```
long gpslat, gpslon;
int fix, gpsvel, gpshead, gpsalt, lowlat, highlat, lowlon, highlon, gpserror, data1_old, data2_old;
const int PWSonar = 8;
const int PWDoppler = 9;
const boolean ON      = LOW;
const boolean OFF     = HIGH;

int largo = 100; // number of characters to read from the ecosounder
char dataeco[200]; // here is where the ecosounder data will be saved
int index = 0; // used to index dataeco
int index2 = 0; // used to index ecodepth and ecospeed
//int index3 = 0;
int counter = 0; // used to count the ',' separator
char ecodepth[5]; // depth data from ecosounder m
char ecospeed[5]; // speed data from ecosounder [km/h]
int foundecodepth = 0; // to stop searching for depth info
int foundecospeed = 0;
float deptheco = 100;
float speedeco = 0;

void setup() {
  fix=0;
  gpsvel=0;
  gpshead=123;
  gpsalt=321;
  lowlat=2;
  highlat=2;
  lowlon=4;
  highlon=4;
  gpserror=5;
  Serial.begin(115200); //Turn on serial monitor
  pinMode(PWSonar, OUTPUT);
  pinMode(PWDoppler, OUTPUT);

  digitalWrite(PWSonar, OFF); // default is to start with the sonar off
  digitalWrite(PWDoppler, OFF); // default is to start with the Doppler Rd off
  GPS.begin(9600); //Turn on GPS at 9600 baud
  GPS.sendCommand("$PGCMD,33,0*6D");  //Turn off antenna update data
  GPS.sendCommand(PMTK_SET_NMEA_OUTPUT_RMCGGA); //Request RMC and GGA Sentences only
  GPS.sendCommand(PMTK_SET_NMEA_UPDATE_5HZ); //Set update rate to 1 hz
  delay(1000);

  clearGPS(); //Clear old and corrupt data from serial port
  data[2] = 1; // default GPS readout
  data1_old = 1;
  data2_old = 1;

  Serial2.begin(4800);

}
```

Pre Processor	Software Name
MEGA2	AXV_Mega2

```
void loop() {
 while(Serial.available()<3) {
  switch (data[2])  {
   case 1:
    readGPS();
    break;
   case 2:
    readSONAR();
    break;
  }
 }
 data[0]=Serial.read();
 data[1]=Serial.read(); // command byte from RPI: turning on/off Doppler Rd
 data[2]=Serial.read(); // command for sonar or GPS reading

 switch (data[1])  {
  case 1:          // turn on sonar turn off doppler
   if (data[1]!=data1_old) {
    digitalWrite(PWDoppler, OFF);
    digitalWrite(PWSonar, ON);
    delay(500);
    //clearSONAR();
   }
   break;
  case 2:         // turn on doppler, turn off sonar
   if (data[1]!=data1_old) {
    digitalWrite(PWSonar, OFF);
    digitalWrite(PWDoppler, ON);
   }
   break;
  case 4:
    if (data[1]!=data1_old) {
    digitalWrite(PWSonar, OFF);
    digitalWrite(PWDoppler, OFF);
   }
   break;
 }
 data1_old = data[1];

 switch (data[2])  {
  case 1:          // gps request
   if (data[2]!=data2_old) {
   clearGPS();readGPS(); }
   printGPS(); clearGPS(); readGPS();
   break;
  case 2:         // sonar request
   if (data[2]!=data2_old) {
   //clearSONAR();
   readSONAR();}
   printSONAR(); //clearSONAR();
   readSONAR();
```

Pre Processor	Software Name
MEGA2	AXV_Mega2

```
    break;
  }
  data2_old = data[2];
}

void  printGPS(){

    fix = (int)GPS.fix;
    gpslat = long(GPS.latitudeDegrees*10000000);
    lowlat = gpslat;
    highlat = gpslat >> 16;
    gpslon = long(GPS.longitudeDegrees*10000000);
    lowlon = gpslon;
    highlon = gpslon >> 16;
    gpserror = int(GPS.fixquality*100); // horizontal dilusion in cm
    gpsalt = int(GPS.altitude*100);
    gpsvel = int(GPS.speed*100);
    gpshead = int(GPS.angle*100);

    Serial.write(90);
    Serial.write(94);
    Serial.write((int)fix);
    Serial.write(highByte(gpserror));
    Serial.write(lowByte(gpserror));
    Serial.write(highByte(highlat));
    Serial.write(lowByte(highlat));
    Serial.write(highByte(lowlat));
    Serial.write(lowByte(lowlat));
    Serial.write(highByte(highlon));
    Serial.write(lowByte(highlon));
    Serial.write(highByte(lowlon));
    Serial.write(lowByte(lowlon));
    Serial.write(highByte(gpsalt));
    Serial.write(lowByte(gpsalt));
    Serial.write(highByte(gpsvel));
    Serial.write(lowByte(gpsvel));
    Serial.write(highByte(gpshead));
    Serial.write(lowByte(gpshead));
    Serial.write(0xFF);

   clearGPS();
}

void readGPS() {
  while(!GPS.newNMEAreceived()) gpsdata=GPS.read();  //Loop until Rx valid NMEA data
  GPS.parse(GPS.lastNMEA()); //Parse NMEA data
  while(!GPS.newNMEAreceived()) gpsdata=GPS.read(); //Loop until you have a good NMEA sentence
  GPS.parse(GPS.lastNMEA()); //Parse NMEA sentence*/
}

void clearGPS() {
```

Pre Processor	Software Name
MEGA2	AXV_Mega2

```
  while(!GPS.newNMEAreceived())gpsdata=GPS.read(); //Loop until Rx valid NMEA data
  GPS.parse(GPS.lastNMEA()); //Parse NMEA sentence
  while(!GPS.newNMEAreceived()) gpsdata=GPS.read(); //Loop until Rx valid NMEA data
  GPS.parse(GPS.lastNMEA()); //Parse NMEA sentence
  while(!GPS.newNMEAreceived()) gpsdata=GPS.read(); //Loop until Rx valid NMEA data
  GPS.parse(GPS.lastNMEA()); //Parse NMEA sentence
}

void readSONAR(){
while(index <=largo){
if(Serial2.available() > 0) {dataeco[index]=(Serial2.read()); index = index +1;}}
if(index>=largo){
  foundecodepth = 0; foundecospeed = 0; index = 0;
  while((index < largo)&&(foundecodepth == 0)){
    if(dataeco[index] == 'S' && dataeco[index+1] == 'D' && dataeco[index+2] == 'D' && dataeco[index+3] == 'B' &&
dataeco[index+4] == 'T'){
      index = index + 4;
      while(index < largo){
        if(dataeco[index] == ','){
           counter = counter + 1;
           if(counter == 3){
             index = index + 1;
             while(dataeco[index+1] != 'M'){
               ecodepth[index2] = dataeco[index];
               index = index + 1;
               index2 = index2 +1;
             }
             deptheco = 100*atof(ecodepth); // in [cm]

           }
       }
       index = index + 1;
     }
     foundecodepth = 1;
   }
     while(index2 <= 4){
       ecodepth[index2]=0;
       index2 = index2 + 1;
     }
   index2 = 0;
   index = index + 1;
  }
  index = 0;
  counter = 0;

  while((index < largo)&&(foundecospeed == 0)){
    if(dataeco[index] == 'V' && dataeco[index+1] == 'W' && dataeco[index+2] == 'V' && dataeco[index+3] == 'H' &&
dataeco[index+4] == 'W'){
      index = index + 4;
      while(index < largo){
```

Pre Processor	Software Name
MEGA2	AXV_Mega2

```
      if(dataeco[index] == ','){
         counter = counter + 1;
         if(counter == 7){
           index = index + 1;
           while(dataeco[index+1] != 'K'){
             ecospeed[index2] = dataeco[index];
             index = index + 1;
             index2 = index2 +1;
           }
           speedeco = atof(ecospeed)*100/3.6; // in [cm/s]
         }
      }
      index = index + 1;
    }
      foundecospeed = 1;
  }
    while(index2 <= 5){
      ecospeed[index2]=(char)0;
      index2 = index2 + 1;
    }
  index2 = 0;
  index = index + 1;
 }
 index = 0;
 counter = 0;
}

}

void clearSONAR(){
  Serial2.end();
  Serial2.begin(4800);
}

void printSONAR(){
     Serial.write(90);
     Serial.write(94);
     Serial.write(highByte(int(deptheco)));
     Serial.write(lowByte(int(deptheco)));
     Serial.write(highByte(int(speedeco)));
     Serial.write(lowByte(int(speedeco)));
     Serial.write(0xFF);
     //clearSONAR();
}
```

Table 30 PID, Tail and SA Administration Program.

Pre Processor	Software Name
TEENSY 3.1C	AXV_PID

```
#include <Wire.h>
#include <Adafruit_PWMServoDriver.h>
#include <PID_v1.h>

Adafruit_PWMServoDriver pwm = Adafruit_PWMServoDriver();

double Setpoint, Setpoint_Depth, Output_land, Output_sea, Head_err, Depth, Servo_cmd, Output_depth, turninplaceSb, turninplacePt;
double aggKpL=4, aggKiL=0.2, aggKdL=1, aggKpS=4, aggKiS=0.2, aggKdS=1; // K = true => agressive, K = false => conservative
double consKpL=1, consKiL=0.01, consKdL=0.25, consKpS=1, consKiS=0.05, consKdS=0.25;
double KpD=1, KiD=0.2, KdD=0.5; // for the servos
int servopos;
int SAact = 0;
int RevFactor = -1; // for going foward = -1, reverse = 1

byte data[10];
int basespeed, basespeedfwd, basespeedaft, roll, pitch;
const int SAdist = 6;
const int SAdepth = 5;
const int MCPwr = 2;
const int SAswIR = 1;
const int thrusterUpDown = 6;// value is 6;
const int thrusterSb = 0;
const int thrusterPt = 1;
const int servoSb = 4;
const int servoPt = 5;
const int landSb = 2; // value is 2
const int landPt = 3; // value is 3
const boolean ON     = LOW;
const boolean OFF    = HIGH;
boolean para = true, MC_status = false; // 'para' is stop in spanish
boolean land = false, sea = false, tail = false, SA = false, Kland = true, Ksea = true, fwd = true, back = false;
int depth = 0;

// ticks values for 50 Hz PRF
float velfactor = 15;
int neutral = 307;//306; //307.2 for 50 Hz. with 307 one of the test motors (with wax) continued fwd 1.5 ms pulse
int maxpwm = 410 ; // 409.6 for 50 Hz 2.0 ms pulse
int minpwm = 205;// 1.0 ms pulse

int neutralservo = 307;//
int maxpwmservo = 430 ; // 2.1 ms
int minpwmservo = 184;// 0.9 ms pulse

PID LandPID(&Head_err, &Output_land, &Setpoint,consKpL,consKiL,consKdL, DIRECT);
PID SeaPID(&Head_err, &Output_sea, &Setpoint,consKpS,consKiS,consKdS, DIRECT);
PID DepthPID(&Depth, &Output_depth, &Setpoint_Depth,KpD,KiD,KdD, DIRECT);
```

Pre Processor	Software Name
TEENSY 3.1C	AXV_PID

```
void setup() {

  pinMode(SAdist, INPUT);
  pinMode(SAdepth, INPUT);
  pinMode(SAswIR, INPUT);
  pinMode(MCPwr, OUTPUT);
  digitalWrite(MCPwr, OFF);

  pinMode(13, OUTPUT);
  digitalWrite(13, HIGH);
  delay(200);
  digitalWrite(13, LOW);
  delay(200);
  digitalWrite(13, HIGH);
  delay(200);
  digitalWrite(13, LOW);


  Head_err = 0;
  Setpoint = 0;
  data[4] = 0;
  turninplaceSb = 0;
  turninplacePt = 0;
  Setpoint_Depth = 0;
  LandPID.SetMode(MANUAL);
  SeaPID.SetMode(MANUAL);
  DepthPID.SetMode(MANUAL);
  LandPID.SetOutputLimits(-30,30);
  SeaPID.SetOutputLimits(-50,50);
  DepthPID.SetOutputLimits(-50,50);
  LandPID.SetSampleTime(100);
  SeaPID.SetSampleTime(300);
  DepthPID.SetSampleTime(300);

  Serial.begin(115200);
  pwm.begin();
  pwm.setPWMFreq(50);
  basespeedfwd = int(neutral + 35);
  basespeedaft = int(neutral - 35);
  basespeed = basespeedfwd;
  pwm.setPWM(landSb,0,neutral);
  pwm.setPWM(landPt,0,neutral);
  pwm.setPWM(thrusterSb,0,neutral);
  pwm.setPWM(thrusterPt,0,neutral);
  pwm.setPWM(thrusterUpDown,0,neutral);
  pwm.setPWM(servoSb,0,neutralservo);
  pwm.setPWM(servoPt,0,neutralservo);
  Output_land = 0;
  Output_sea = 0;
  Output_depth = 0;
}
```

Pre Processor	Software Name
TEENSY 3.1C	AXV_PID

```
void loop() {
  while(Serial.available()<7){

    LandPID.Compute();

    if((tail == 0)&&(data[4] == 2)){          // code that tells "turn in place"
      turninplaceSb = RevFactor*(Output_land);
      turninplacePt = -RevFactor*(Output_land);
    }
    else{
      turninplaceSb = 0;
      turninplacePt = 0;
    }

    if(SA == true && (digitalRead(SAdist)||digitalRead(SAswIR))) {
      pwm.setPWM(landSb,0,neutral + turninplaceSb);
      pwm.setPWM(landPt,0,neutral + turninplacePt);
      //LandPID.SetMode(MANUAL);
      digitalWrite(13, LOW);
      SAact = 1;
      //para = true;
    }
    else SAact = 0;
    if(SA == true && sea == true && (digitalRead(SAdepth))){
      pwm.setPWM(thrusterUpDown,0,minpwm+velfactor); // 85% speed up!
      delay(3000);
      SAact = 1;
    }
    if(land == true && para == false && SAact == 0){
      pwm.setPWM(landSb,0,basespeed + RevFactor*(Output_land));
      pwm.setPWM(landPt,0,basespeed - RevFactor*(Output_land));
      }
    if(sea == true && para == false){
      SeaPID.Compute();
      pwm.setPWM(thrusterSb,0,basespeed + (Output_sea));
      pwm.setPWM(thrusterPt,0,basespeed - (Output_sea));
      DepthPID.Compute();
      pwm.setPWM(thrusterPt,0,Output_depth);
      SAact = 0;
      }
    }

  data[0]=Serial.read(); // mesage heading
  data[1]=Serial.read(); // head low
  data[2]=Serial.read(); // head high
  data[3]=Serial.read(); // servo error low
  data[4]=Serial.read(); // servo error high
  data[5]=Serial.read(); // cmddue1: 1 = land, 2 = tail, 4 = sea, 8 = stop, 16 = direction, 32 = SA, 64 = K conservative
or aggressive, 128 = Ksea
  data[6]=Serial.read(); // Depth in [cm]
```

Pre Processor	Software Name
TEENSY 3.1C	AXV_PID

```
// ***************general settings*****************************

  Head_err = long(data[1] + data[2]*256);
  if(Head_err>32768) Head_err = long(Head_err - 65535); // to handle negative number (2 complement)
  if(abs(Head_err)<2) Head_err = 0;

  if(data[5] & byte(8)) {
    para = true; // means stop
    if(land){
      //LandPID.SetMode(MANUAL);
      pwm.setPWM(landSb,0,neutral + turninplaceSb);
      pwm.setPWM(landPt,0,neutral + turninplacePt);
      digitalWrite(13, LOW);
    }
    if(sea){
      pwm.setPWM(thrusterSb,0,neutral);
      pwm.setPWM(thrusterPt,0,neutral);
    }
  }
  else {
    if(para == true) { // means that it was stop before
      //LandPID.SetMode(AUTOMATIC);
      digitalWrite(13, HIGH);
    }
    para = false;
  }

  if(byte(data[5]) & byte(32)){  // SA activated
    if(SA == false){
      SA = true;
    }
  }
  else {
    if(SA == true){
      SA = false;
    }
  }

// **********************land PID*********************************

  if(data[5] & byte(1)){   // land PID request
    land = true;
    if(!MC_status){            // if true, means that MC is OFF
      LandPID.SetMode(AUTOMATIC); // turn ON PID control
      digitalWrite(MCPwr, ON);
      MC_status = true;
      pwm.setPWM(landSb,0,neutral);
      pwm.setPWM(landPt,0,neutral);
      pwm.setPWM(servoSb,0,neutral); // set the servos to 0 [deg] position
      pwm.setPWM(servoPt,0,neutral);
```

Pre Processor	Software Name
TEENSY 3.1C	AXV_PID

```
  }
  if(data[5] & byte(2)){ // tail request
   tail = true;
   Servo_cmd = long(data[3] + data[4]*256);
   if(Servo_cmd>32768) Servo_cmd = long(Servo_cmd - 65535);
   servopos = map(Servo_cmd, 90, 180, minpwmservo, maxpwmservo);
   pwm.setPWM(servoSb,0,servopos); // set the servos to the angle beta
   pwm.setPWM(servoPt,0,servopos);
  }

  if(data[5] % byte(16)){ // means foward direction
   if(fwd == false){
    fwd = true;
    back = false;
    basespeed = basespeedfwd;
    RevFactor = -1;
   }
  }
  else {                  // means is backwards direction
   if(back == false){
    fwd = false;
    back = true;
    basespeed = basespeedaft;
    RevFactor = 1;
   }
  }
  if(data[5] % byte(64)){   // means conservative parameters
   if(Kland == false)  {       // then the parameters must change
    Kland = true;
    LandPID.SetTunings(consKpL,consKiL,consKdL);
   }
  }
  else {                   // means aggressive parameters
   if(Kland == true)  {     // then the parameters must change
    Kland = false;
    LandPID.SetTunings(aggKpL,aggKiL,aggKdL);
  }
  }
 }                         // end of active land commands
 else {                    // land request == 0
  if(MC_status == true){     // if true, means that MC is ON => motors controlles must be turn off
   LandPID.SetMode(MANUAL);
   pwm.setPWM(landSb,0,neutral);
   pwm.setPWM(landPt,0,neutral);
   digitalWrite(13, LOW);
   MC_status = false;
   digitalWrite(MCPwr, OFF);
  }
  land = 0;
 }
// **********sea PID*******************************
```

Pre Processor	Software Name
TEENSY 3.1C	AXV_PID

```
if(data[5] & byte(4)){   // sea PID request
   Depth = data[6];
   if(!sea) {
    SeaPID.SetMode(AUTOMATIC); // turn on sea PID
    DepthPID.SetMode(AUTOMATIC); // turn on up down PID
    sea = true;
   }
   if(para){
    SeaPID.SetMode(MANUAL);
    pwm.setPWM(thrusterSb,0,neutral);
    pwm.setPWM(thrusterPt,0,neutral);
    sea = false;
   }
   if(data[5] % byte(16)){ // means foward direction
    if(fwd == false){
     fwd = true;
     back = false;
     basespeed = basespeedfwd;
     SeaPID.SetControllerDirection(DIRECT);
    }
   }
   else {                  // means is backwards direction
    if(back == false){
     fwd = false;
     back = true;
     basespeed = basespeedaft;
     SeaPID.SetControllerDirection(REVERSE);
    }
   }
   if(data[5] % byte(128)){   // means conservative parameters
    if(Ksea == false)  {      // then the parameters must change
     Ksea = true;
     SeaPID.SetTunings(consKpS,consKiS,consKdS);
    }
   }
   else {                   // means aggressive parameters
    if(Ksea == true)  {     // then the parameters must change
     Ksea = false;
     SeaPID.SetTunings(aggKpS,aggKiS,aggKdS);
   }
   }
 }                         // end of active sea commands
 else {                    // sea request == 0
  sea = false;
 }
 Serial.write(90); Serial.write(95);  // ID Teensy 3.1C
 Serial.write(lowByte(int(Output_land)));                                    Serial.write(lowByte(int(Output_sea)));
Serial.write(lowByte(int(Output_depth))); Serial.write(lowByte(SAact)); Serial.write(177);

 if(data[0] == 255){    // To reset PID Output to zero
```

Pre Processor	Software Name
TEENSY 3.1C	AXV_PID

```
  LandPID.SetOutputLimits(0,0);
  SeaPID.SetOutputLimits(0,0);
  DepthPID.SetOutputLimits(0,0);
  delay(150);
  LandPID.Compute();
  setup();
 }

 if(data[0] == 200){    // To do ESC's thrusters calibration
  LandPID.SetOutputLimits(0,0);
  SeaPID.SetOutputLimits(0,0);
  DepthPID.SetOutputLimits(0,0);
  delay(150);
  LandPID.Compute();
  setup();
  digitalWrite(13, HIGH);
  pwm.setPWM(thrusterSb,0,maxpwm);
  pwm.setPWM(thrusterPt,0,maxpwm);
  pwm.setPWM(thrusterUpDown,0,maxpwm);
  delay(8000);
  pwm.setPWM(thrusterSb,0,neutral);
  pwm.setPWM(thrusterPt,0,neutral);
  pwm.setPWM(thrusterUpDown,0,neutral);
  delay(5000);
  pwm.setPWM(thrusterSb,0,minpwm);
  pwm.setPWM(thrusterPt,0,minpwm);
  pwm.setPWM(thrusterUpDown,0,minpwm);
  delay(5000);
  pwm.setPWM(thrusterSb,0,neutral);
  pwm.setPWM(thrusterPt,0,neutral);
  pwm.setPWM(thrusterUpDown,0,neutral);
  delay(2000);
  setup();
 }
 if(data[0] == 210){    // for  MC calibration
  LandPID.SetOutputLimits(0,0);
  SeaPID.SetOutputLimits(0,0);
  DepthPID.SetOutputLimits(0,0);
  delay(150);
  LandPID.Compute();
  LandPID.SetMode(MANUAL);
  SeaPID.SetMode(MANUAL);
  DepthPID.SetMode(MANUAL);
  digitalWrite(MCPwr, ON);
  MC_status = true;
  delay(15000);
  pwm.setPWM(landSb,0,neutral);
  pwm.setPWM(landPt,0,neutral);
  digitalWrite(13, HIGH);
  pwm.setPWM(landSb,0,maxpwm);
  pwm.setPWM(landPt,0,maxpwm);
```

Pre Processor	Software Name
TEENSY 3.1C	AXV_PID

```
    delay(8000);
    pwm.setPWM(landSb,0,neutral);
    pwm.setPWM(landPt,0,neutral);
    delay(5000);
    pwm.setPWM(landSb,0,minpwm);
    pwm.setPWM(landPt,0,minpwm);
    delay(5000);
    pwm.setPWM(landSb,0,neutral);
    pwm.setPWM(landPt,0,neutral);
    delay(20000);
    setup();
  }
}
```

Table 31 IMU Software.

Pre Processor	Software Name
TEENSY 3.1A	AXV_TeenA1

```
//  This file is part of RTIMULib-Teensy
//
//  Copyright (c) 2014-2015, richards-tech
//
//  Permission is hereby granted, free of charge, to any person obtaining a copy of
//  this software and associated documentation files (the "Software"), to deal in
//  the Software without restriction, including without limitation the rights to use,
//  copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the
//  Software, and to permit persons to whom the Software is furnished to do so,
//  subject to the following conditions:
//
//  The above copyright notice and this permission notice shall be included in all
//  copies or substantial portions of the Software.
//
//  THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED,
//  INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
//  PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT
//  HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION
//  OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE
//  SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

#include <Wire.h>
#include <SD.h>
#include <SPI.h>
#include <EEPROM.h>
#include "I2Cdev.h"
```

Pre Processor	Software Name
TEENSY 3.1A	AXV_TeenA1

```
#include "RTIMULib.h"

RTIMU *imu;                                          // the IMU object
RTIMUSettings *settings;                             // the settings object

#define  SERIAL_PORT_SPEED  115200

unsigned long lastDisplay;
unsigned long lastRate;
int sampleCount;
byte data[4];
float deltat = 0;
unsigned long t1=0;
unsigned long t2=0;
int led = 0;

void setup()
{
   int errcode;
   Serial.begin(SERIAL_PORT_SPEED);
   while (!Serial) {
      ; // wait for serial port to connect.
   }
   Wire.begin();
   settings = new RTIMUSettings();
   imu = RTIMU::createIMU(settings);                        // create the imu object
   Serial.print("TeensyIMU starting using device ");
   Serial.println(imu->IMUName());
   if ((errcode = imu->IMUInit()) < 0) {  Serial.print("Failed to init IMU: "); Serial.println(errcode);   }

   if (imu->getCompassCalibrationValid())
      Serial.println("Using compass calibration");
   else
      Serial.println("No valid compass calibration data");

   // set up any fusion parameters here
   imu->setSlerpPower(0.02);
   imu->setGyroEnable(true);
   imu->setAccelEnable(true);
   imu->setCompassEnable(true);

   pinMode(13, OUTPUT);
   digitalWrite(13, LOW);
   lastDisplay = lastRate = millis();
   sampleCount = 0;     }

void loop()
{
   unsigned long now = millis();
   unsigned long delta;
   RTIMU_DATA imuData;
```

Pre Processor	Software Name
TEENSY 3.1A	AXV_TeenA1

```
  if (imu->IMURead()) {                    // get the latest data if ready yet
    imuData = imu->getIMUData();
      if (!imu->IMUGyroBiasValid()) {

// Most of the rest of the code had been modified from here:
        digitalWrite(13, LOW);
        led = 0;
        //Serial.println("calculating gyro bias");
       }
       else if(led == 0) {
        digitalWrite(13, HIGH);
        led = 1;
       }

if (!(Serial.available()<3)) {
       //lastDisplay = now;
       data[0]=Serial.read();
       data[1]=Serial.read();
       data[2]=Serial.read();
       Serial.print(RTMath::displayDegreesAVX("Pose:", imuData.fusionPose)); // fused output
}  }  }
```

Table 32 RTIMU Library.

Pre Processor	Software Name
TEENSY 3.1A	RT IMU Library
Available on line in: https://github.com/richards-tech/RTIMULib-Teensy	

Table 33 RTIMU Library Modifications for MOSARt.

RTIMU Library files to be modified	
Software Name	**Modification**
RTMath.cpp	On line 50 add:

```
int ax, ay, az; // angles on each axis
const char *RTMath::displayDegreesAVX(const char *label, RTVector3& vec) // modified 22 ABR to fit AXV format
{
   az = int(100*vec.z() * RTMATH_RAD_TO_DEGREE);
   ax = int(100*vec.x() * RTMATH_RAD_TO_DEGREE);
   ay = int(100*vec.y() * RTMATH_RAD_TO_DEGREE);
   Serial.write(90); Serial.write(91);
```

RTIMU Library files to be modified	

Serial.write(highByte(301)); Serial.write(lowByte(301));
Serial.write(highByte(az)); Serial.write(lowByte(az));
Serial.write(highByte(ax)); Serial.write(lowByte(ax));
Serial.write(highByte(ay)); Serial.write(lowByte(ay));
Serial.write(highByte(100)); Serial.write(lowByte(100)); // dummy velocity
Serial.write((190)); Serial.write(lowByte(190)); // dummy
}	

Software Name	**Modification**
RTMath.h	On line 54 add:

// added 23 May: inside "public". These are display routines
static const char *displayDegreesTest(const char *label, RTVector3& vec);
static const char *displayDegreesAVX(const char *label, RTVector3& vec);	

Software Name	**Modification**
RTIMUsettings.cpp	On line 416 modify:
m_GD20HM303DLHCGyroSampleRate = L3GD20H_SAMPLERATE_100; // 23 MAY: changed from 50 to 100	

Table 34 Doppler Radar Velocity Measurement.

Pre Processor	Software Name
TEENSY 3.1B	AXV_TeenB

```
#include <FreqMeasure.h>

float vel, velavg, cosalpha;
int n;
float frequency;
elapsedMicros deltat;
byte data[3];

void setup() {
  FreqMeasure.begin();
  vel = 0; frequency = 0; cosalpha = 0.7071; n = 0; deltat = 0; // cosalpha set to 45 deg
}

void printinfo() {Serial.write(100); Serial.write(lowByte(int(velavg))); Serial.write(lowByte(int(vel))); Serial.write(170);}
// velocity in [cm/s] it should not go above 254 cm/s (1 byte). It will output also raw velocity

void loop() {
  if(deltat>=20000){ // 20ms has a cut off velocity of 10 cm/s
    if(FreqMeasure.available()) {frequency = FreqMeasure.countToFrequency(FreqMeasure.read());}
    else {frequency = 0;} // if there is no measurement means that there is no movement
    vel = float(frequency*3e8/(2*10.525e9*cosalpha)); // calculation of the velocity with the antenna placed -45 deg
elevation
    /*if((vel <= 1.5)){ */ // filter to validate velocity. If not valid, it will request another velocity inmidiattly  &&(vel >= 0.09)
      velavg = velavg + vel; // calculates average velocity with filtered velocity
      n = n + 1;
      deltat = 0;
```

Pre Processor	Software Name
TEENSY 3.1B	AXV_TeenB

```
    //}
  }
  if(Serial.available()>=2) {
    data[0] = Serial.read();
    data[1] = Serial.read();
    velavg = (100*float(velavg/n));
    vel = (100*vel);
    printinfo();
    vel = 0;
    velavg = 0;
    n = 0;

    }
}
```

C. MATLAB PROGRAMS

Only those programs that are not being directly replicated on the MP/ PP are presented on this section. Programs which only objective was to display data or plots are also omitted.

Table 35 Magnetic Calibration

Software Name	Function
AXV_MagCal.m	From raw magnetometer data, it performs soft and hard iron compensation in XY plane.

```
clc
clear all
clf

rawmag2 = fopen('magnetometer6.txt','r');
rawmagm2 = transpose(fscanf(rawmag2,'%f %f %f',[3 Inf] ));
fclose(rawmag2);

Xmax = max(rawmagm2(:,1));
Ymax = max(rawmagm2(:,2));
Xmin = min(rawmagm2(:,1));
Ymin = min(rawmagm2(:,2));

Xoff = (Xmax + Xmin)/2;
Yoff = (Ymax + Ymin)/2;

subplot(221)
plot(rawmagm2(:,1),rawmagm2(:,2),'r.')
hold on
plot(0,0,'k+')
```

Software Name	Function
AXV_MagCal.m	From raw magnetometer data, it performs soft and hard iron compensation in XY plane.

```
hold off
grid on
grid minor
xlabel('X [\muT]','FontSize', 18)
ylabel('Y [\muT]','FontSize', 18)
title('a. Magnetometer Uncalibrated Data - XY Plane','FontSize', 18)
axis equal

subplot(222)
plot(rawmagm2(:,1) - Xoff, rawmagm2(:,2) - Yoff,'r.')
hold on
plot(0,0,'k+')
grid on
grid minor
xlabel('X [\muT]','FontSize', 18)
ylabel('Y [\muT]','FontSize', 18)
title('b. Magnetometer Hard-Iron calibrated Data - XY Plane','FontSize', 18)
axis equal

for a=1:length(rawmagm2(:,2))
    radius(a) = sqrt((rawmagm2(a,1) - Xoff).^2 + (rawmagm2(a,2) - Yoff).^2);
end

index = find(radius== max(radius));
i = index(1);
omega = 90 - atan2d((rawmagm2(i,2) - Yoff),(rawmagm2(i,1) - Xoff))
plot(rawmagm2(i,1) - Xoff, rawmagm2(i,2) - Yoff,'bx')

hold off

subplot(223)
for a=1:length(rawmagm2(:,2))
    xrot(a) = (rawmagm2(a,1) - Xoff).*cosd(omega) - (rawmagm2(a,2) - Yoff).*sind(omega);
    yrot(a) = (rawmagm2(a,1) - Xoff).*sind(omega) + (rawmagm2(a,2) - Yoff).*cosd(omega);
    rotrad(a) = sqrt(xrot(a).^2 + yrot(a).^2);
    theta(a) = atan2d(yrot(a),xrot(a));
end

plot(xrot,yrot,'r.')
hold on
index = find(rotrad== max(rotrad));
i = index(1);
plot(xrot(i), yrot(i),'kx')

b=2;
posmax(1)=0;
for a=1:length(rawmagm2(:,2))
    if(abs(theta(a))<= (183) && abs(theta(a))>= (177))
        posmax(b)=rotrad(a);
        if(posmax(b)<posmax(b-1))
            posmax(b)=posmax(b-1)
            index = a
        end
        b = b + 1;
    end
end

plot(xrot(index), yrot(index),'k*')
plot(0,0,'k+')
hold off
grid on
grid minor
```

Software Name	Function
AXV_MagCal.m	From raw magnetometer data, it performs soft and hard iron compensation in XY plane.

```
xlabel('X [\muT]','FontSize', 18)
ylabel('Y [\muT]','FontSize', 18)
title('c. Magnetometer Data Rotated - XY Plane','FontSize', 18)
axis equal

mayor_axis = rotrad(i)
minor_axis = posmax(b-1)
scale_factor = minor_axis/mayor_axis
yrot = yrot*scale_factor;

subplot(224)
for a=1:length(rawmagm2(:,2))
    x(a) = (xrot(a)).*cosd(-omega) - (yrot(a)).*sind(-omega);
    y(a) = (xrot(a)).*sind(-omega) + (yrot(a)).*cosd(-omega);
end
plot(x, y, 'b.')
hold on
plot(0,0,'k+')
hold off
grid on
grid minor
axis equal
xlabel('X [\muT]','FontSize', 18)
ylabel('Y [\muT]','FontSize', 18)
title('d. Final Magnetometer Calibrated Data - XY Plane','FontSize', 18)
```

Table 36 Magnetic Tilt Compensation

Software Name	Function
AXV_MagTilt.m	It performs roll and pitch compensation to XY magnetometer data.

```
clc
clear all
clf

rawmag2 = fopen('magacc4.TXT','r');
rawmagm2 = transpose(fscanf(rawmag2,'%f %f %f %f %f %f',[6 Inf]));
fclose(rawmag2);

Mx_raw = rawmagm2(:,1);
My_raw = rawmagm2(:,2);
Mz_raw = rawmagm2(:,3);
roll = rawmagm2(:,4);
pitch = rawmagm2(:,5);
Factory_head = rawmagm2(:,6);
```

Software Name	Function
AXV_MagTilt.m	It performs roll and pitch compensation to XY magnetometer data.

```
for a=1:length(rawmagm2(:,1))
    mx(a) = Mx_raw(a).*cosd(pitch(a)) + Mz_raw(a).*sind(pitch(a));
    my(a) = Mx_raw(a).*sind(pitch(a)).*sind(roll(a)) + My_raw(a).*cosd(roll(a)) -
Mz_raw(a).*sind(roll(a)).*cosd(pitch(a));
end

head = atan2d(-my, mx);
for a=1:length(head)
    if (head(a) <=-180)
        head(a) = head(a)+360;
    end
end

headun = atan2d(-My_raw(:),Mx_raw(:));
for a=1:length(headun)
    if (headun(a) <=-180)
        headun(a) = headun(a)+360;
    end
end

subplot(2,1,1)
plot(head)
hold all
plot(headun)
legend('Tilt Corrected Heading','Raw Heading')
xlabel('Time Stamp','FontSize', 18)
ylabel('Heading [Deg]','FontSize', 18)
title('a. Tilt Compensated Heading','FontSize', 18)
grid on
grid minor
% plot(Factory_head)
hold off

subplot(2,1,2)
plot(roll,'.')
hold all
plot(pitch,'.')
legend('Roll', 'Pitch')
xlabel('Time Stamp','FontSize', 18)
ylabel('Degrees','FontSize', 18)
title('b. Roll and Pitch','FontSize', 18)
hold off
grid on
grid minor
```

Table 37 Compensated Filter for IMU

Software Name	Function
AXV_IMUcomp.m	It performs data fusion for the accelerometer, gyroscope and magnetometer.

```
% AXV TESIS - COMPLEMENTARY FILTER TEST
% File writen by Oscar Garcia 24/08/15
clear all
clc
clf

imu = fopen('imucomp2.txt','r');
```

Software Name	Function
AXV_IMUcomp.m	It performs data fusion for the accelerometer, gyroscope and magnetometer.

```
imum = transpose(fscanf(imu,'%f %f %f %f %f %f %f',[7 Inf] ));
fclose(imu);

time(1)= imum(1,7);

for a=2:length(imum(:,7))
    time(a) = imum(a,7) + time(a-1);
end
time = time/1000;
% subplot(221)
plot(time,imum(:,1),'b.')
hold on
%plot(time,imum(:,4),'b.')

angleg(1) = imum(1,1);
for a=2:length(imum(:,4))
    angleg(a) = angleg(a-1) + imum(a,4);
end

plot(time,angleg,'m.','LineWidth',1)

alpha = 0.97;
angle(1)=imum(1,1);

for a=2:length(imum(:,7));
    angle(a) = (alpha*(angle(a-1) + imum(a,4)) + (1.0 - alpha)*imum(a,1));
end

plot(time,angle,'k','LineWidth',2)

grid on
grid minor
xlabel('Time [s]','FontSize', 18)
ylabel('Angle [deg]','FontSize', 18)
title('Data Fusion (Roll) - Complementary Filter','FontSize', 18)
legend('Accelerometer', 'Gyroscope', 'Complementary Filter Fusion')
```

Table 38 IMU First Order Kalman Filter

Software Name	Function
AXV_KalmanIMU.m	It performs IMU data fusion and filtering using a first order Kalman filter.

```
% AXV TESIS - IMU KALMAN FILTER TEST
% File writen by Oscar Garcia 30/08/15
% Adaptation from Kalman Filter for Beginners, Phil Kim

clear all
clc
clf

imu = fopen('imucomp2.txt','r');
imum = transpose(fscanf(imu,'%f %f %f %f %f %f %f',[7 Inf] ));
fclose(imu);

time(1)= imum(1,7);
```

Software Name	Function
AXV_KalmanIMU.m	It performs IMU data fusion and filtering using a first order Kalman filter.

```
for a=2:length(imum(:,7))
    time(a) = imum(a,7) + time(a-1);
end
time = time/1000;


Nsamples = length(imum(:,7));
EulerSaved = zeros(Nsamples, 3);

dt = 1; % dt is already incorporated on the arduino output

for k=1:Nsamples
    p = imum(k,4)/(57.2957795*imum(k,7));
    q = imum(k,5)/(57.2957795*imum(k,7));
    r = imum(k,6)/(57.2957795*imum(k,7));
    A = eye(4) + imum(k,7)*(1/2)*  [0 -p -q -r;
                                    p 0 r -q;
                                    q -r 0 p;
                                    r q -p 0];

    phi =   imum(k,1)/57.2957795;
    theta = imum(k,2)/57.2957795;
    psi =   imum(k,3)/57.2957795;
    z = EulerToQuaternion(phi, theta, psi);

    [phi theta psi] = EulerKalman(A, z);
    EulerSaved(k,:) =  [phi theta psi];
end

phiDeg = EulerSaved(:,1)*57.2957795;
thetaDeg = EulerSaved(:,2)*57.2957795;
psiDeg = EulerSaved(:,3)*57.2957795;

% hold on
plot(time,imum(:,1),'b.','LineWidth',2)
hold on
plot(time,phiDeg(:,1),'r','LineWidth',2)
hold off
grid on
grid minor
xlabel('Time [s]','FontSize', 18)
ylabel('Angle [deg]','FontSize', 18)
title('Data Fusion (Roll) - Kalman Filter','FontSize', 18)
legend('Accelerometer', 'Kalman Filter Fusion')
```

Table 39 IMU Extender Kalman Filter

Software Name	Function
AXV_IMU_EKF.m	It performs IMU data fusion and filtering using an extended Kalman filter.

```
% AXV TESIS - IMU EKF TEST
% File writen by Oscar Garcia 30/08/15
% Adaptation from Kalman Filter for Beginners, Phil Kim

clear all
clc
```

Software Name	Function
AXV_IMU_EKF.m	It performs IMU data fusion and filtering using an extended Kalman filter.

```
% clf

imu = fopen('imucomp2.txt','r');
imum = transpose(fscanf(imu,'%f %f %f %f %f %f %f',[7 Inf] ));
fclose(imu);

time(1)= imum(1,7);

for a=2:length(imum(:,7))
    time(a) = imum(a,7) + time(a-1);
end
time = time/1000;


Nsamples = length(imum(:,7));
EulerSaved = zeros(Nsamples, 3);

dt = 1; % dt is already incorporated on the arduino output

for k=1:Nsamples
    dt = imum(k,7);
    p = imum(k,4)/(57.2957795*imum(k,7));
    q = imum(k,5)/(57.2957795*imum(k,7));
    r = imum(k,6)/(57.2957795*imum(k,7));
    phi =   imum(k,1)/57.2957795;
    theta = imum(k,2)/57.2957795;
    psi =   imum(k,3)/57.2957795;

    [phi theta psi] = EulerEKF([phi theta]', [p q r], dt);
    EulerSaved(k,:) =  [phi theta psi];
end

phiDeg = EulerSaved(:,1)*57.2957795;
thetaDeg = EulerSaved(:,2)*57.2957795;
psiDeg = EulerSaved(:,3)*57.2957795;

% hold on
plot(time,imum(:,1),'b.','LineWidth',2)
hold on
plot(time,phiDeg(:,1),'r','LineWidth',2)
hold off
grid on
grid minor
xlabel('Time [s]','FontSize', 18)
ylabel('Angle [deg]','FontSize', 18)
title('Data Fusion (Roll) - EKF','FontSize', 18)
% title('Data Fusion (Roll) - Complementary vs EKF','FontSize', 18)
legend('Accelerometer', 'EKF Fusion')
% legend('Raw Accelerometer','Complementary Filter','EKF')
```

Table 40 Euler to Quaternion Function

Software Name	Function
EulerToQuaternion.m	It performs transformation from Euler angles to Quaternion form.

```
function z = EulerToQuaternion(phi, theta, psi)
%
%
sinPhi = sin(phi/2); cosPhi = cos(phi/2);
sinTheta = sin(theta/2); cosTheta = cos(theta/2);
sinPsi = sin(psi/2); cosPsi = cos(psi/2);

z = [cosPhi*cosTheta*cosPsi + sinPhi*sinTheta*sinPsi; sinPhi*cosTheta*cosPsi -
cosPhi*sinTheta*sinPsi;
    cosPhi*sinTheta*cosPsi + sinPhi*cosTheta*sinPsi; cosPhi*cosTheta*sinPsi -
sinPhi*sinTheta*cosPsi; ];
```

Table 41 First Order Kalman Filter

Software Name	Function
EulerKalman.m	It performs Kalman filtering (1^{st} order) using quaternions as input and outputs the results in Euler angles.

```
% AXV TESIS - KALMAN FILTER TEST - IMU
% File writen by Oscar Garcia 28/08/15
% REFERENCE: Kalman Filter for Beginners, Phil Kim

function [phi theta psi] = EulerKalman(A, z)
%
%
persistent H Q R
persistent x P
persistent firstRun

if isempty(firstRun)
    H = eye(4);
    Q = 0.001*eye(4);
    R = 0.05*eye(4);
    x = z;%[1 0 0 0]';
    P = 1*eye(4);
    firstRun = 1;
end

xp = A*x;
Pp = A*P*A' + Q;

K = Pp*H'*inv(H*Pp*H' + R);

x = xp + K*(z - H*xp);
P = Pp - K*H*Pp;

phi = atan2(2*(x(3)*x(4) + x(1)*x(2)), 1 - 2*(x(2)^2 + x(3)^2));
theta = -asin(2*(x(2)*x(4) - x(1)*x(3)));
psi = atan2(2*(x(2)*x(3) + x(1)*x(4)), 1 - 2*(x(4)^2 + x(3)^2));
```

Table 42 Extended Kalman Filter

Software Name	Function
EulerEKF.m	It performs Kalman filtering (2nd order) using Euler angles as inputs.

```
% AXV TESIS - EKF TEST - IMU
% File writen by Oscar Garcia 28/08/15
% REFERENCE: Kalman Filter for Beginners, Phil Kim

function [phi theta psi] = EulerEKF(z, rates, dt)

persistent H Q R
persistent x P
persistent firstRun

if isempty(firstRun)
    qu = 0.0003;
    H =[1 0 0; 0 1 0];
    Q = [qu 0 0; 0 qu 0; 0 0 0.1];
    R = 0.08*[1 0; 0 1];
    x = [0 0 0]';
    P = 10*eye(3);
    firstRun = 1;
end

A = Ajacob(x,rates, dt);

xp = fx(x, rates, dt);
Pp = A*P*A' + Q;

K = Pp*H'*inv(H*Pp*H' + R);

x = xp + K*(z - H*xp);
P = Pp - K*H*Pp;

phi = x(1);
theta = x(2);
psi = x(3);

function xp = fx(xhat, rates, dt)

phi = xhat(1);
theta = xhat(2);

p = rates(1);
q = rates(2);
r = rates(3);

xdot = zeros(3,1);
xdot(1) = p + q*sin(phi)*tan(theta)+r*cos(phi)*tan(theta);
xdot(2) = q*cos(phi)-r*sin(phi);
xdot(3) = q*sin(phi)*sec(theta)+r*cos(phi)*sec(theta);

xp = xhat + xdot*dt;

function A = Ajacob(xhat, rates, dt)

A = zeros(3,3);

phi = xhat(1);
theta = xhat(2);

p = rates(1);
q = rates(2);
r = rates(3);
```

Software Name	Function
EulerEKF.m	It performs Kalman filtering (2nd order) using Euler angles as inputs.

```
A(1,1) = q*cos(phi)*tan(theta)-r*sin(phi)*tan(theta);
A(1,2) = q*sin(phi)*sec(theta)^2+r*cos(phi)*sec(theta)^2;
A(1,3) = 0;

A(2,1) = -q*sin(phi) - r*cos(phi);
A(2,2) = 0;
A(2,3) = 0;

A(3,1) = q*cos(phi)*sec(theta) - r*sin(phi)*sec(theta);
A(3,2) = q*sin(phi)*sec(theta)*tan(theta) + r*cos(phi)*sec(theta)*tan(theta);
A(3,3) = 0;

A= eye(3) + A*dt;
```

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF REFERENCES

[1] U.S. Department of Defence, *Unmanned Systems Integrated Roadmap FY2013 – 2038*. Washington D.C.: DOD, 2013.

[2] M. Ariza, *The Design and Implementation of a Prototype Surf zone Robot for Waterborne Operations*. Monterey, CA: NPS, 2015.

[3] T. Bell, *Sea-Shore Interface Robotic Design*. Monterey, CA: NPS, 2014.

[4] J. Hickle and S. Halle, *The Design And Implementation of a Semi-Autonomous Surf-Zone Robot Using Advanced Sensors and a Common Robot Operating System*. Monterey, CA: NPS, 2011.

[5] CH Robotics*, AN-1008 – Sensors for Orientation Estimation*. Payson, UT, 2012

[6] M. Pedley, *Tilt Sensing Using a Three-Axis Accelerometer*. Freescale Semiconductor, Inc., Austin, TX, 2013.

[7] ST Microelectronics, *LSM303DLHC Ultra Compact High Performance e-Compass 3D Accelerometer and 3D magnetometer Module Datasheet*. Santa Clara, CA: ST Microelectronics, 2011.

[8] ST Microelectronics, *TA0343 Technical Article: Everything About ST Microelectronics' 3-Axis digital MEMS Gyroscopes.* Santa Clara, CA: ST Microelectronics, 2011.

[9] M. Ferraina, *AN4506 Application Note – L3GD20H: 3-Axis Digital Output Gyroscope*. Santa Clara, CA: ST Microelectronics, 2015.

[10] W. Storr. (2015). *Electronic Tutorials: Hall Effect Sensor.* [Online]. Available: http://www.electronics-tutorials.ws/electromagnetism/hall-effect.html

[11] L. Ada. (2015). *Adafruit Ultimate GPS*. Adafruit Learning System. [Online]. Available: https://learn.adafruit.com/adafruit-ultimate-gps

[12] GlobalTop Technology Inc., *FGPMMOPA6C GPS Standalone Module Data Sheet (Revision V0A)*. Taiwan: GlobalTop Technology Inc., 2011.

[13] M. Yuen, *Dilution of Precision (DOP) Calculation for Mission Planning Purposes*. Monterey, CA: NPS, 2009.

[14] A. El-Rabbany, *Introduction to GPS, The Global Positioning System*. London: Artech House, 2002.

[15] A. Pirti, *Multipath and Multipath Reduction in the Obstructed Areas by Using Enhanced Strobe Correlator (ESC) Tecnique*. Tehnicki Vjesnik. 2015.

[16] G. Baddeley. (2001). *GPS-NMEA Sentence Information.* [Online]. Available: http://aprs.gids.nl/nmea/

[17] L. Diaz and C. Granell, J. Huerta, *Discovery of Geospatial Resources: Methodologies, Technologies and Emergent Applications.* Hershey: Information Science Reference, 2012.

[18] A. Schneider. (2015). GPS Visualizer. [Online]. Available: http://www.gpsvisualizer.com/map_input?form=data

[19] Measurement Specialties, *MS5803-14BA Miniature 14 bar Module Datasheet*. Freemont: Measurement Specialties, 2012.

[20] Agilsense, *HB100 10.525 GHz Microwave Motion Sensor Module Version 1.02*. Jurong: ST Electronics.

[21] Agilsense, *MSAN-001 X-Band Microwave Motion Sensor Module Application Note Version 1.02*. Jurong: ST Electronics.

[22] Maxbotic, *HRXL-MaxSonar – WR Series Datasheet*. Brainerd, MN: MaxBotic, 2006.

[23] R. Siegwart, I. Nourbakhsh, D. Scaramuzza, *Introduction to Autonomous Mobile Robots (2nd Edition)*. Massachusetts: MIT Press, 2004.

[24] J. Fraden, *Handbook of Modern Sensors* (4th ed.). New York, NY: Springer, 2010.

[25] T. Bonar. (2015). *Using Multiple MaxSonar Sensors.* [Online]. Available: http://www.maxbotix.com/articles/031.htm

[26] S. Monk, *Programming Arduino: Next Steps, Going Further with Sketches*. New York: Mc Graw Hill, 2014.

[27] K. Valavanis, *Advances in Unmanned Aerial Vehicles – State of the Art and the Road to Autonomy*. Dordrecht: Springer, 2007.

[28] M. Euston, P. Coote, R. Mahony, J. Kim and T. Hamel, *A Complementary Filter for Attitude Estimation of a Fixed-Wing UAV*, Nice, IEE, 2008.

[29] L. Gear, *My Complementary Filter Tutorial*. Austin, TX, National Instruments, 2012.

[30] R, Mahony, T. Hamel, J. Pflimlin, *Complementary Filter Design on the Special Orthogonal Group SO (3)*. Seville: IEEE, 2005.

[31] P. Kim, *Kalman Filter for Beginners with Matlab Examples*. Korea: A-JIN, 2010.

[32] MIT OCW, *Chapter 9: Kinematics of Moving Frames*. Massachusetts Institute of Technology: MIT OpenCourseWare, http://ocw.mit.edu (Accessed June 2015). License: Creative Commons BY-NC-SA.

[33] P. Kim, *Rigid Body Dynamics for Beginners: Euler angles & Quaternions*. Korea: A-JIN, 2013.

[34] B. Beauregard. (2012). *Improving the Beginner's PID*. [Online]. Available: http://brettbeauregard.com/blog/2011/04/improving-the-beginners-pid-introduction/

[35] R. Harkins, *PC4015 – Advanced Physics Lab*. Monterey, CA: NPS, 2014.

[36] *Eigen library for Teensy 3.1 forum.* (2015). [Online]. Available: https://forum.pjrc.com/threads/28181-Eigen-library-for-linear-algebra-Teensy-3-1-Help-needed/page2.

[37] S. Tzafestas, *Introduction to Mobile Robot Control*. Waltham: Elsevier, 2014.

[38] J. Borenstein, Y. Koren, *Real-Time Obstacle Avoidance for Fast Mobile Robots*. Virginia: IEEE, 1988.

[39] H. Choset, K. Lynch, S. Hutchinson, G. Kantor, W. Burgard, L. Kacraki and S. Thrun, *Principle of Robot Motion: Theory, Algorithms and Implementations*. London: A Bradford Book, 2005.

[40] CrustCrawler Robotics. (2015). *400HFS-L Hi-Flow Thruster Product Guide and Waranty*. [Online]. Available: http://crustcrawler.com/index.php

[41] RFbeam Microwave GmbH. (2015). *K-LC1a_V4 Doppler Transceiver*. [Online]. Available: http://www.rfbeam.ch/downloads/data-sheets/

THIS PAGE INTENTIONALLY LEFT BLANK

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center
   Ft. Belvoir, Virginia

2. Dudley Knox Library
   Naval Postgraduate School
   Monterey, California